



UNIVERSITÀ  
DEGLI STUDI DI BARI  
ALDO MORO

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA  
INFORMATICA MAGISTRALE

TESI DI LAUREA MAGISTRALE IN  
SISTEMI PER LA COLLABORAZIONE IN RETE

---

Partecipazione assistita di soggetti non  
udenti a eventi pubblici mediante  
trascrizione real-time

---

**Relatori:**

Prof. Filippo Lanubile  
Prof. Fabio Calefato

**Laureando:**

Giuseppe Iaffaldano

**Correlatore:**

Dott. Fabrizio Lippolis

## Indice

1. Introduzione .....	4
2. Cenni teorici .....	5
2.1. Il riconoscimento vocale .....	5
2.2. Il Rispeakeraggio .....	6
2.3. Disabilità uditive.....	8
2.4. Informatici Senza Frontiere - ISF.....	11
3. Analisi dello stato dell'arte .....	15
3.1. Soluzioni per la trascrizione automatica real-time: AVA (ex Transcense) .....	15
3.2. Librerie per lo Speech-To-Text.....	16
3.2.1. Fase 1: Ricerca.....	16
3.2.2. Fase 2: Selezione .....	28
3.2.3. Fase 3: Adozione .....	30
4. Metodologia.....	31
4.1. Metodologia di sviluppo Agile .....	31
4.2. La metodologia Scrum .....	35
4.3. Utilizzo di Scrum.....	38
4.3.1. Controllo versione e GitHub .....	39
4.3.2. ZenHub .....	41
5. Sprint 0 - Preliminare .....	43
5.1. Analisi del contesto lavorativo.....	43
5.2. Analisi degli utenti attesi.....	43
5.3. Analisi degli scenari.....	46
5.4. User Story.....	48
5.5. Casi d'uso .....	51
6. Sprint 1 - Scelta delle Tecnologie .....	56
5.1. Lato Server: NodeJS (6.3.1) .....	56
6.2. Persistenza: MongoDB - Una soluzione NoSQL .....	58
5.3. Relatore e Partecipanti - Socket.IO per il broadcasting .....	61
6.4. Applicazione Web - Cordova per lo sviluppo cross-platform.....	62
5.5. Web Client - Angular 2: Un framework Mobile e Desktop .....	64
6.6. App ibride e Performance - Ionic 2: Un framework omnicomprensivo.....	66
7. Sprint 2 – Primo spike di ASR e Broadcast .....	68
7.1. Primo test Google Web Speech API .....	68
7.2. Architettura del sistema .....	73

7.3. Primo Broadcast di messaggi .....	75
8. Sprint 3 - Prima interfaccia Web e Mobile.....	78
8.1. Comunicazione Server - Android .....	78
8.1.1. Intel XDK.....	78
8.2. Login e Registrazione utente .....	82
9. Sprint 4 - Gestione dati: Utenti ed Eventi .....	103
9.1. Gestione Profilo Utente .....	103
9.1.1. Update.....	103
9.1.2. Delete (Logout) .....	109
9.2. Gestione Eventi.....	111
9.2.1. Create.....	112
9.2.2. Read .....	118
9.2.3. Update.....	123
9.2.4. Delete.....	130
9.3. Gestione Sessioni .....	133
9.3.1. Create.....	133
9.3.2. Read .....	137
9.3.3. Update.....	141
9.3.4. Delete.....	144
9.4. Gestione Interventi .....	146
9.4.1. Create.....	147
9.4.2. Read .....	153
9.4.3. Update.....	156
9.4.4. Delete.....	160
10. Sprint 5 - Completamento del Core-Goal .....	165
10.1. Osservazione Eventi .....	165
10.2. Avvio Eventi.....	171
10.3. Ascolto - Rilettura Eventi .....	180
10.4. Interventi dei partecipanti – Domande al relatore.....	189
10.4.1. Database .....	189
10.4.2. Server Node .....	189
10.4.3. Server Ionic .....	190
10.4.4. Test.....	191
11. Sprint 6 - Deployment multi-piattaforma .....	193
11.1. Deployment del server Node .....	193

11.1.1. Installazione del database MongoDB .....	193
11.1.2. Installazione di NodeJS.....	194
11.1.3. Installazione del server realizzato.....	195
11.2. Deployment del server Ionic .....	195
11.2.1. Applicazione Web Desktop .....	195
11.2.2. Applicazione Android .....	195
11.2.3. Applicazione iOS.....	196
11.3. Predisposizione per il deployment esterno .....	197
11.4. Deployment esterno – Docker .....	199
12. Sprint 7 - Valutazione.....	203
12.1. Fase 1: Accuratezza del riconoscimento.....	203
12.2. Fase 2: Prestazioni del sistema .....	206
13. Conclusioni e Sviluppi Futuri.....	208

# 1. Introduzione

Dagli albori del nuovo millennio, siamo stati spettatori di un avanzamento tecnologico non indifferente che ha sancito la nascita della ormai tanto discussa generazione dei “nativi digitali” ovvero persone abituate sin dalla nascita ad essere supportati dalle tecnologie e dagli strumenti digitali in generale.

Tanto in fretta sono stati pressoché completamente colmati i gap generazionali, grazie agli studi in ambito UX e HCI, ma altrettanto in fretta sono sorti nuove esigenze in ambito di “accessibilità”, ovvero è in aumento l’esigenza di sviluppare tecnologia che permetta davvero a tutti di vivere la propria vita quotidiana senza preoccuparsi delle proprie difficoltà e disabilità.

Si pensi in particolare a chi soffre di deficit uditivi o totale sordità, tali soggetti tendono a sentirsi isolati dal mondo esterno e continuano ad isolarsi per non mostrarsi “inferiori” rispetto agli altri, rinunciando quindi alle interazioni sociali anche più semplici.

Il paradosso che si sta vivendo è quello che vede una tecnologia che si evolve verso il “social” per unire persone, ma che riesce a tenerle distanti nell’interazione reale interpersonale. Quello che ci si auspica è di riuscire davvero a creare tecnologie che possano rendere le persone più vicine tra di loro, non solo virtualmente ma soprattutto fisicamente.

Un’esigenza sempre esistita ma fin ora insoddisfacibile è quella di rendere partecipi soggetti non udenti o con deficit uditivi ad eventi sociali quali conferenze, seminari e lezioni accademiche.

L’obiettivo di questo lavoro di tesi è quello di progettare e realizzare un sistema che permetta, grazie a strumenti di cui ormai tutti sono dotati, e un minimo sforzo, di rendere i suddetti ambienti accessibili dai soggetti citati, progettando un software per il “rispeakeraggio” automatico su dispositivi mobili che possa permettere a tali individui di leggere, piuttosto che ascoltare, quello che il proprio interlocutore sta dicendo.

## 2. Cenni teorici

In questo capitolo saranno illustrati sommariamente i problemi da affrontare dal punto di vista teorico, in modo da individuare il contesto in cui si deve agire e proporre una soluzione adeguata ai problemi effettivi da affrontare.

### 2.1. Il riconoscimento vocale

Il riconoscimento vocale è il processo mediante il quale il linguaggio orale umano viene riconosciuto attraverso una macchina. Sistemi di riconoscimento vocale vengono utilizzati per lo più in applicazioni vocali automatizzate, nella telefonia come per i call center automatici, per sistemi di dettatura, che consentono di dettare discorsi al computer senza dover digitare del testo, oppure per sistemi di controllo di dispositivi di ogni tipo tramite comandi vocali.

Il primo sistema di riconoscimento vocale vide la luce nel 1952, consisteva in un dispositivo per il riconoscimento di singole cifre parlate<sup>1</sup>; un altro dispositivo simile era il prodotto di casa IBM: Shoebox, esposto al Salone di New York del 1964<sup>2</sup>.

In Italia, pionieri della sintesi sono stati i ricercatori del CSELT (Centro Studi E Laboratori Telecomunicazioni) di Torino negli anni Settanta. Nel 1990 il CSELT pubblicò il primo libro italiano sullo stato dell'arte (di cui esiste anche la versione inglese, ristampata nel 2013 da Springer). Successivamente, a seguito della privatizzazione di Telecom Italia, il centro divenne “Loquendo SpA”, maggiore esponente italiano nel settore.

Nel mondo della telefonia mobile, il sistema Loquendo esordisce nel 2003 con il Nokia 6630, destinato ormai a diventare dominante nel 2010 nei diversi smartphone.

I sistemi di riconoscimento vocale adibiti alla scrittura del testo dettato, permettono di dettare documenti in qualunque editor di testo e permettono anche di costruire macro vocali che permettono di eseguire azioni abituali in maniera più rapida.

L'installazione dei software adibiti allo scopo, richiede la lettura di un brano davanti al microfono, per addestrare il programma a riconoscere la voce, che viene registrata e analizzata per costruire una libreria di file vocali (modello vocale dell'utente) al fine di ridurre drasticamente gli errori legati al riconoscimento vocale.

In un secondo momento, il programma chiede un elenco di documenti scritti dall'utente, per memorizzare il suo lessico.

---

<sup>1</sup> Davies, K.H., Biddulph, R. and Balashek, S. (1952) Automatic Speech Recognition of Spoken Digits, *J. Acoust. Soc. Am.* 24(6) pp.637 - 642

<sup>2</sup> IBM Shoebox: [http://www-03.ibm.com/ibm/history/exhibits/specialprod1/specialprod1\\_7.html](http://www-03.ibm.com/ibm/history/exhibits/specialprod1/specialprod1_7.html)

Il W3C<sup>3</sup> ha definito, negli anni, degli standard per le tecnologie vocali, tra cui si annoverano il VoiceXML<sup>4</sup> (alla versione 3 nel 2009) e il CCXML<sup>5</sup>. Per la specifica di grammatiche vocali ha introdotto Speech Recognition Grammar Specification<sup>6</sup>, per la sintesi vocale SSML<sup>7</sup> (Speech Synthesis Markup Language), per la pronuncia PLS<sup>8</sup> (Pronunciation Lexicon Specification), per l'interpretazione semantica dei risultati SISR<sup>9</sup> (Semantic Interpretation for Speech Recognition).

## 2.2. Il Rispeakeraggio<sup>10</sup>

Il termine “rispeakeraggio” è stato proposto per la prima volta da Eugeni e Mack nel 2006<sup>11</sup> come traduzione italiana del più conosciuto termine inglese “respeaking”, che letteralmente significa “riparlare”.

Grazie al diffondersi di questa tecnica e al successo ottenuto nella “Prima giornata di studi internazionale sulla sottotitolazione in tempo reale” dell'università degli studi di Bologna, l'uso di questo termine e della sua versione originale si è diffuso in maniera esponenziale

Quanto a una prima definizione, Eugeni definisce il rispeakeraggio come

*“una riformulazione, una traduzione o una trascrizione di un testo [...] prodotta dal rispeaker ed elaborata dal computer in contemporanea con la produzione del testo di partenza [...]. [Un] software di riconoscimento del parlato procede alla trasformazione dell'input orale in testo scritto.”*

Il rispeakeraggio automatico è dunque il processo attraverso cui un sistema ascolta un testo prodotto oralmente, lo riconosce e lo trasforma in testo scritto.

L'attività della maggior parte dei riconoscitori del parlato si può suddividere in fasi fondamentali:

1. **registrazione** del suono;
2. **riconoscimento** dei singoli enunciati da elaborare: il sistema deve stabilire il punto di inizio e di fine di ogni singola parola.

---

<sup>3</sup> Sito web W3C: <https://www.w3.org/>

<sup>4</sup> W3C VoiceXML (2009): <https://www.w3.org/TR/2009/WD-voicexml30-20090602/>

<sup>5</sup> W3C CCXML (2011): <https://www.w3.org/TR/ccxml/>

<sup>6</sup> W3c Speech Recognition Grammar Specification (2004): <https://www.w3.org/TR/speech-grammar/>

<sup>7</sup> W3C SSML (2004): <https://www.w3.org/TR/speech-synthesis/>

<sup>8</sup> W3c PLS (2008): <https://www.w3.org/TR/2008/REC-pronunciation-lexicon-20081014/>

<sup>9</sup> W3c SISR (2007): <https://www.w3.org/TR/semantic-interpretation/>

<sup>10</sup> I contenuti sono estratti da C. Eugeni (Napoli 2008), Tesi di Dottorato: “La sottotitolazione in diretta TV. Analisi strategica del rispeakeraggio Verbatim di BBC News” - Relatore: Prof.ssa R.M. Bollettieri, Coordinatore del Dottorato: Prof.ssa G. Di Martino

<sup>11</sup> Eugeni, C. (2006) “Introduzione al rispeakeraggio televisivo”. In Eugeni, C. and G. Mack (a cura di) Proceedings of the first international seminar on real-time intralingual subtitling. InTRALinea special issue: respeaking. [www.intralinea.it](http://www.intralinea.it)

Esistono essenzialmente due modalità di riconoscimento: i sistemi basati sul riconoscimento di pattern confrontano il parlato con dei pattern noti o appresi determinando così delle corrispondenze; i sistemi basati sulla fonetica acustica sfruttano invece conoscenze sul corpo umano per confrontare feature del parlato tra loro (proprietà fonetiche come il suono delle vocali o delle sillabe). La maggior parte dei sistemi moderni utilizza l'approccio di riconoscimento di pattern perché si adatta molto bene alle tecniche computazionali esistenti e tende a raggiungere prestazioni migliori in termini di accuratezza.

3. **pre-filtraggio** (pre-amplificazione, normalizzazione, spostamento di banda, ecc.). I metodi di pre-filtraggio più comuni sono il metodo ‘Banco di Filtri’, che usa una serie di filtri audio per preparare il campione audio, e la ‘Codifica Lineare Predittiva’ che usa una funzione di predizione per calcolare gli scostamenti dalla pronuncia standard di una parola.
4. **suddivisione dei dati** in un formato ‘pulito’, utilizzabile nella fase successiva di elaborazione dell’input;
5. eventuale **ulteriore filtraggio** di ciascun dato (frame o banda di frequenze) durante il quale si effettuano gli ultimi aggiustamenti del campione prima delle fasi di confronto e matching. Spesso si fanno operazioni di allineamento temporale e normalizzazione;
6. **confronto** con le possibili combinazioni tra fonemi e grafemi e matching dell’input con l’enunciato corrispondente. La maggior parte delle tecniche utilizzate per attuare questa operazione è basata sul confronto del frame corrente con dei campioni noti.
7. **trascrizione.**

Per funzionare al meglio, ogni software per il riconoscimento del parlato impone dei vincoli in base al tipo di enunciato target.

A seconda della natura dell’enunciato richiesta si distinguono:

- **enunciati isolati** (o parlato discreto): il software richiede che ciascun elemento da riconoscere (parola, sintagma o breve frase) termini con un periodo di pausa (assenza di segnale audio, su entrambi i lati della finestra di campionamento, cioè sia prima, sia dopo l’enunciazione) per permettere al sistema di elaborare l’enunciato appena ascoltato;
- **enunciati connessi**: un’evoluzione del precedente, che permette la sovrapposizione della fase di dettatura di un enunciato con l’elaborazione dell’enunciato precedente da parte del sistema. L’utente non deve quindi aspettare che venga elaborato l’enunciato precedente, ma deve comunque scandire bene le parole e le pause tra una parola e l’altra;

- **parlato continuo:** il sistema utilizza particolari tecniche per determinare i confini di un enunciato. Sistemi di riconoscimento basati su questa tecnica permettono all'utente di parlare in maniera quasi del tutto naturale. Sono di questo tipo i sistemi ad oggi più utilizzati per dettare un testo a una macchina;
- **parlato spontaneo:** il sistema è in grado di elaborare un testo che sembra naturale e non preparato e quindi di riconoscere il parlato spontaneo. Il software, oltre a determinare i confini di un enunciato, distingue anche le parole dagli elementi non lessicali che possono essere tipici della produzione di un testo orale non preparato e che normalmente riducono la qualità del riconoscimento. Attualmente questo tipo di sistemi è in continua evoluzione e non sono ancora accurati al 100% per lingue diverse dall'inglese.

Tutti i tipi di riconoscimento del parlato appena descritti possono essere dipendenti o indipendenti da chi parla (rispettivamente speaker dependent e speaker independent). I primi sono progettati per soddisfare le esigenze di uno specifico utente; generalmente, presentano un'elevata accuratezza quando utilizzati dallo stesso utente, ma hanno prestazioni inferiori se usati nello stesso contesto da utenti differenti senza cambiare il profilo vocale. Tali sistemi assumono, infine, che l'utente non modifichi significativamente timbro e ritmo d'eloquio. Al contrario, i sistemi indipendenti sono progettati per essere usati da utenti diversi. Questi sistemi, definiti anche adattivi, di solito funzionano in una prima fase come sistemi indipendenti e poi, utilizzando tecniche di addestramento, si adattano al singolo utente per migliorare la qualità del riconoscimento.

## 2.3. Disabilità uditiva<sup>12</sup>

Attraverso l'udito l'essere umano assorbe i valori della cultura, riceve informazioni, messaggi emotivi, stabilisce rapporti sociali. L'udito rappresenta dunque un importante strumento per lo sviluppo intellettuale. Il termine "sordità" identifica una perdita di funzionalità importante, che comporta, molto spesso, problemi nello sviluppo del linguaggio.

L'OMS (Organizzazione Mondiale della Sanità) definisce il soggetto ipoacusico come colui

*"la cui acuità uditiva non è sufficiente a permettergli di imparare la sua lingua, di partecipare alle normali attività della sua età, di seguire con profitto l'insegnamento scolastico generale".*

---

<sup>12</sup> I contenuti sono estratti da "Handicap e psicologia: La sordità" - A cura della Dott.ssa Katia Carlini, Presidente dell'Associazione Psicologia in Movimento

Generalmente, la sordità è la conseguenza di patologie genetiche (50%) o di problemi prenatali. In una percentuale più bassa può essere la conseguenza di prematurità o sofferenza perinatale (20%), oppure acquisita durante l'infanzia per cause infettive, tossiche o traumatiche.

Le conseguenze di tali patologie sono un deficit sensoriale che può essere più o meno grave, a seconda della profondità e del periodo di insorgenza della sordità. Si parla infatti di:

- sordità totale o cofosi, quando il deficit è superiore a 85 dB;
- ipoacusia profonda quando il deficit è compreso tra i 60 e gli 85 dB;
- ipoacusia lieve quando il deficit è compreso tra i 40 e i 60 dB;
- sordastro, quando il deficit è inferiore a 40 dB.

Secondo l'INPS (Istituto Nazionale della Previdenza Sociale), sono considerati sordi i minorati sensoriali dell'udito affetti da sordità congenita o acquisita durante l'età evolutiva (fino a 12 anni) che abbia impedito il normale apprendimento del linguaggio parlato, purché la sordità non sia di natura esclusivamente psichica o dipendente da cause di guerra, di lavoro o di servizio.

Si considera causa impeditiva del normale apprendimento del linguaggio parlato l'ipoacusia (pari o superiore a 75 decibel di HTL - pure-tone hearing threshold di media tra le frequenze 500, 1000, 2000 Hz nell'orecchio migliore) che renda o abbia reso difficoltoso tale normale apprendimento.

Da un punto di vista anatomo-fisiologico la sordità può essere di:

- **trasmissione** in cui la conduzione ossea è normale, così come la percezione della parola;
- **ricezione** in cui si evidenziano difficoltà notevoli nell'apprendimento fonetico e la ricezione delle parole è abbastanza deformata tanto da apparire isolate o associate alle precedenti;
- **alterazione dell'identificazione** in cui l'integrazione uditiva e/o la simbolizzazione al livello centrale risultano alterate.

La comparsa di una sordità profonda fin dalla nascita priva il bambino di una fonte di informazione che gli avrebbe permesso di scoprire il mondo in un'interazione circolare e di acquisire l'uso del linguaggio. Di fatto, il bambino completamente sordo che fisiologicamente utilizza il balbettio intorno ai 2-3 mesi, cessa quasi del tutto di emettere i suoni già a partire dal 5-6 mese. Nel caso dell'ipoacusia lieve e del sordastro, invece, l'acquisizione del linguaggio è ancora possibile ma mentre nel primo caso l'articolazione e la voce resteranno difettose, nel secondo caso il linguaggio potrà seguire un normale sviluppo.

In caso di diagnosi precoce, purtroppo poco frequente, di piccolo deficit uditivo, ci sarebbero buone possibilità, attraverso una valida educazione logopedica e una buona protesizzazione,

di apprendere correttamente l'uso del linguaggio. Nella maggior parte dei casi, invece, i bambini con lievi difficoltà uditive non vengono riconosciuti immediatamente. Così da non poter essere inseriti per tempo in adeguati programmi rieducativi, fondamentali per il raggiungimento di una normale integrazione sociale e scolastica.

Un primo esame, il test delle Emissioni Otoacustiche, che testa la funzionalità dell'orecchio interno, può essere effettuato già dal secondo, terzo giorno di vita. Un altro test, che può essere compiuto successivamente e comunque durante il primo anno di vita, è l'esame dei potenziali uditivi (ABR). Troppo frequentemente però, la diagnosi di sordità, non viene fatta prima dei due anni, età in cui il linguaggio ha il suo massimo sviluppo.

L'apprendimento del bambino sordo avviene attraverso la vista, che sostituisce l'udito e con adeguate scelte metodologiche: lettura labiale e/o Lingua dei Segni (LIS).

Al di fuori di qualsiasi lesione specifica, legata ad una particolare eziologia, lo sviluppo intellettivo del bambino sordo mostra un'intelligenza pratica vicino alla norma. Tuttavia, dalla valutazione delle capacità intellettive con prove non verbali e con test concepiti in maniera adatta, si evince un deficit nell'ambito dell'astrazione e del pensiero formale. Alcune difficoltà si presentano, inoltre, nelle prove di memoria visiva e di orientamento temporo-spatiale. Di fatto, la deprivazione uditiva non può essere rimpiazzata dall'insieme delle informazioni provenienti dalla vista. Tuttavia, questo tipo di ritardo tende a colmarsi con l'età, fino a scomparire del tutto a partire dai 15-16 anni.

La profondità della sordità, così come il suo esordio precoce, comporta molto spesso difficoltà emotive, alle quali genitori ed educatori devono porre attenzione. La consapevolezza di non essere in grado di utilizzare la comunicazione verbale al pari di fratelli e amici può originare un senso di esclusione nel bambino sordo. Il sordo ha una percezione del proprio corpo come imperfetto e malato che tende a sfogare manifestando mancanza di disciplina ed emotività assai labile. La disfunzione, inoltre, accresce il legame di dipendenza e di protezione tra il bambino e i genitori. Le conseguenze da un punto di vista psicologico sono le difficoltà a relazionarsi con i coetanei, a comprendere e a condividere i sentimenti degli altri. Il comportamento è infatti piuttosto egocentrico o comunque ripiegato su sé stesso. Nei casi in cui la sordità non è troppo grave, il bambino in grado di parlare e di udire è maggiormente inserito nel contesto relazionale, anche se non sarà comunque capace di conoscere completamente gli stati emotivi degli altri, essendo inadeguato a riconoscere le inflessioni della voce o i giochi di parole.

La sordità, che insorge in un'età più tardiva, comporta un importante vissuto depressivo, con la necessaria conseguenza di dover elaborare il lutto per la grave perdita subita. Di fatto, in queste situazioni, il soggetto sperimenta uno stato regressivo, di sofferenza. In ogni caso, la stragrande maggioranza degli studi è d'accordo nel riconoscere l'estrema importanza delle

reazioni dell’ambiente esterno per l’equilibrio psicofisico del soggetto sordo. In un’ottica biopsicosociale, appunto, devono essere considerate al primo posto la salute e le potenzialità dell’individuo, mentre la sua disabilità solo in relazione ai limiti che essa pone nei confronti delle normali attività del soggetto. La finalità non è, infatti, una diagnosi ghettizzante ma lo spunto per individuare le esigenze del particolare deficit uditivo e superare, per quanto possibile, i limiti delle restrizioni alla partecipazione sociale. Si tratta in altri termini di offrire al sordo una migliore qualità della vita attraverso adeguati servizi di telecomunicazione quali DTS<sup>13</sup> o Videotelefonino<sup>14</sup>; di informazione come spazi televisivi mediante la Lingua dei Segni, interprete, sottotitolazione; di mobilità tramite segnali luminosi per emergenze; di istruzione e formazione e di lavoro. Inoltre, non va dimenticato l’impiego massiccio delle moderne tecnologie con i loro relativi servizi integrati oramai in quasi tutti gli attuali smartphone. Esse hanno permesso un’integrazione “naturale” del soggetto con problemi uditivi nell’ambiente circostante.

Un discorso a parte merita la LIS o Lingua dei Segni Italiana. Si tratta di una vera e propria lingua con caratteristiche grammaticali proprie. Così, se a uno sguardo profano la comunicazione LIS può apparire un semplice gesticolare, solo la grande attenzione visiva del debole di udito percepisce una corrispondenza precisa di significati nei segni prodotti da una o entrambe le mani del suo interlocutore.

## 2.4. Informatici Senza Frontiere - ISF

Informatici Senza Frontiere ONLUS nasce alla fine del 2005, quando un gruppo di informatici italiani decide di utilizzare le proprie conoscenze per contribuire a colmare il divario digitale, in Italia e negli altri Paesi.

L’organizzazione no-profit si prefigge l’obiettivo di utilizzare le proprie conoscenze (informatiche e non) per fornire un aiuto concreto a chi vive in una situazione di povertà e di emarginazione o come mezzo per offrire delle possibilità in più di inserimento sociale alle categorie disagiate.

Dopo dieci anni di attività, Informatici Senza Frontiere vanta oggi dieci sezioni regionali con più di 300 membri, uomini e donne, che stanno contribuendo agli obiettivi dell’associazione attraverso la creazione di corsi, l’informatizzazione di ospedali rurali e di centri di accoglienza, la creazione di programmi specifici e di reti informatiche, la collaborazione con

---

<sup>13</sup> Dispositivo Telefonico per Sordi: un telefono alternativo alla vocalità, poiché dotato di una tastiera con la quale si può scrivere un messaggio a un altro utente munito del medesimo apparecchio.

<sup>14</sup> Il Videotelefonino permette di vedere la persona contattata telefonicamente mediante una telecamera e un computer e di interagire mediante la lettura labiale e/o la LIS.

le scuole, le università e le carceri e mediante la realizzazione di applicazioni dirette nel mondo della disabilità.

Comprendendo l'importanza del contributo di Informatici Senza Frontiere, le Nazioni Unite nel maggio 2013 hanno invitato l'associazione a presentare una parte del suo lavoro a Ginevra, durante il Vertice Mondiale sulla Società dell'Informazione Forum 2013 ITU, riconoscendola come realtà rappresentante a livello europeo delle potenzialità dell'IT nel difficile campo della disabilità.

Informatici Senza Frontiere spesso opera in modo indipendente, ma il più delle volte, insieme ad altre ONG o organizzazioni no-profit e diverse università con una lunga storia di tesi sperimentali (Bari, Pisa, Venezia, Padova, Torino, Milano), considerata la trasversalità della disciplina informatica rispetto alle altre. Di progetto in progetto ISF è diventata nel tempo il "braccio informatico" di tante realtà del mondo del volontariato.

Si consideri che Informatici Senza Frontiere è fermamente convinta che avere accesso alle tecnologie dell'informazione e della comunicazione sia prerequisito essenziale per lo sviluppo sociale ed economico; le tecnologie dell'informazione dovrebbero essere considerate beni essenziali perché capaci di semplificare procedure, velocizzare contatti, aprire mercati, mettere in comunicazione in pochi minuti migliaia di persone accorciando le distanze che le separano.

In definitiva Informatici Senza Frontiere è la realizzazione, nella sua massima espressione, della comunità internazionale che si mobilita per combattere il digital divide e garantire la forma più genuina di democrazia, ossia quella in cui ogni cittadino può essere parte attiva della vita della propria comunità ed avere a portata di mano tutto il sapere umano, assieme alla possibilità di arricchirlo.

Tali valori, fondamentali, sono recepiti sia nello Statuto che nel Regolamento dell'Associazione.

*"L'Associazione non ha fini di lucro, né diretti né indiretti. L'Associazione è apartitica e aconfessionale. Essa perseguità esclusivamente finalità di solidarietà sociale, in ambito sia nazionale che internazionale, attraverso la realizzazione di progetti in cui le competenze nelle tecnologie informatiche e nelle comunicazioni possano contribuire al superamento di realtà discriminatorie ed emarginanti, nonché al miglioramento della qualità della vita delle popolazioni indigenti nei Paesi in via di sviluppo e delle categorie sociali più deboli nei Paesi sviluppati."*

*(Art. 2 dello Statuto di ISF)*

*“Tutte le attività svolte nell’ambito dei progetti ISF sono effettuate dai soci e da eventuali simpatizzanti a titolo gratuito, salvo diversa indicazione scritta del Consiglio Direttivo per specifici progetti nazionali finanziati, fermo restando il divieto ex lege di remunerare i soci dell’Associazione.”*

*(Regolamento di ISF)*

Tra i progetti principali di Informatici Senza Frontiere, si annoverano I.S.A. – I Speak Again<sup>15</sup>, un comunicatore gratuito mediante sintesi vocale, disponibile in versione offline installabile o in versione web application. I.S.A. è Open Source, i sorgenti del sistema<sup>16</sup> e dei plugin<sup>17</sup> sono disponibili su GitHub; Interfaccia I.M.A. – I Move Again<sup>18</sup>, Frutto di un lavoro di tesi di laurea che hanno visto collaborare ISF ed il prof. Filippo Lanubile del Dipartimento di Informatica dell’Università degli Studi di Bari, I.M.A. è un’estensione di I.S.A. per pazienti con gravi disabilità motorie, che funge da interfaccia per sedia a rotelle motorizzata; Paperboy/Strillone<sup>19</sup>, applicazione progettata per smartphone, tablet e PC desktop, che consente alle persone non vedenti o con gravi problemi di visione di sfogliare e ascoltare le notizie di interesse del proprio quotidiano preferito mediante la sintesi vocale integrata, utilizzando semplicemente i quattro angoli dello schermo. Strillone è disponibile come web application<sup>20</sup>, come app su Google Play, App Store e su Windows Phone Store, il codice open source della versione web è rilasciato con licenza GPL su GitHub<sup>21</sup>. Strillone è stata finalista al premio mondiale dell’ITU, agenzia delle telecomunicazioni delle Nazioni Unite ed è presente nei documenti di Outcome<sup>22</sup> dell’evento WSIS, tenutosi a Ginevra nel 2013 e nel documento di stocktaking<sup>23</sup> dell’intero 2014, come best practice adottata in Italia.

Nell’ambito artistico, i progetti principali sono “Musical Instruments for Persons with Disabilities” in collaborazione con l’associazione AccordiAbili, presieduta dal maestro Vincenzo Deluci, Informatici Senza Frontiere ha messo a disposizione il proprio know-how per progettare e realizzare strumenti musicali per persone con disabilità. Il primo progetto ha dato vita alla tromba elettromeccanica ELMECII, suonata dallo stesso maestro Deluci<sup>24</sup>;

---

<sup>15</sup> Web application: <http://www.ispeakagain.org>

<sup>16</sup> Sorgenti di I.S.A.: <https://github.com/informatici/isa>

<sup>17</sup> Sorgenti dei plugin di I.S.A.: <https://github.com/informatici/isa-firefox-addon>

<sup>18</sup> Video I.M.A.: <https://www.youtube.com/watch?v=AkenVzgbCiM>

<sup>19</sup> Video Strillone: <https://www.youtube.com/watch?v=zk-PnPs1VKg>

<sup>20</sup> Web application Strillone: <http://www.walks.to/strillone>

<sup>21</sup> Sorgente di Strillone: <https://github.com/informatici/strillone>

<sup>22</sup> Documento di Outcome WSIS: [http://www.itu.int/net/wsis/implementation/2013/forum/inc/doc/outcome/S-POL-WSIS.OD\\_FORUM-2013-PDF-E.pdf](http://www.itu.int/net/wsis/implementation/2013/forum/inc/doc/outcome/S-POL-WSIS.OD_FORUM-2013-PDF-E.pdf)

<sup>23</sup> Documento di stocktaking: [http://www.itu.int/dms\\_pub/itu-s/opb/pol/S-POL-WSIS.REP-2014-PDF-E.pdf](http://www.itu.int/dms_pub/itu-s/opb/pol/S-POL-WSIS.REP-2014-PDF-E.pdf)

<sup>24</sup> Il maestro Deluci suona la sua ELMECII: <https://www.youtube.com/watch?v=s-yHQXmSuIY>

Sensoltre<sup>25</sup>, il primo percorso multisensoriale al buio tra quadri tattili (Giovanni Pedote in arte Giope, a cura di Emanuela Ferri) con l'ausilio della tecnologia NFC (comunicazione in prossimità).

---

<sup>25</sup> Sito della mostra Sensoltre: <http://www.sensoltre.org>

### 3. Analisi dello stato dell'arte

Questo capitolo è dedicato alla presentazione delle soluzioni già esistenti per la trascrizione simultanea e ai servizi disponibili per il riconoscimento del parlato continuo. Seguirà una sezione dedicata alla selezione del servizio da utilizzare per la soluzione proposta in questo lavoro di tesi. La selezione è stata effettuata in tre fasi consecutive rispettivamente ricerca, selezione, adozione. Nella fase di ricerca sono state ricercati tutti i servizi che potrebbero adattarsi alle esigenze, nella fase di selezione sono stati esclusi quei servizi ancora non maturi e inadeguati alle esigenze e nella fase di adozione sono stati invece testati grossolanamente i servizi selezionati in precedenza che meglio si adattavano al contesto lavorativo per verificare che fossero realmente adeguati per lo sviluppo della soluzione proposta.

#### 3.1. Soluzioni per la trascrizione automatica real-time: AVA (ex Transcense)

Sono diversi i servizi commerciali che propongono soluzioni per la sottotitolazione in tempo reale di programmi televisivi, notiziari, conferenze, seminari e per la sbobinatura; tuttavia tutti i servizi individuati si basano sull'intervento di un operatore umano che si occupa di ridettare alla macchina quello che ascolta o che si occupa di correggere quello che i sistemi automatici riescono a trascrivere. Il dispendio di risorse umane allo scopo di questo lavoro di tesi risulta superfluo e alquanto anacronistico. L'unico servizio promettente nasce come soluzione innovativa per la reintroduzione di soggetti audiolesi in contesti sociali usuali; tale servizio è AVA (ex Transcense).

Il progetto Transcense nasce a San Francisco (USA) nel 2014 come campagna di crowdfunding sulla piattaforma Indiegogo<sup>26</sup>, chiedendo fondi per un ammontare di 25.000\$, alla chiusura della campagna sono stati raggiunti 43.559\$.

L'obiettivo del progetto è di includere persone non udenti in contesti sociali di ogni tipo permettendogli di “ascoltare” le conversazioni leggendole senza quindi il vincolo da parte dell'interlocutore di apprendere la lingua dei segni.

Il video di presentazione del progetto illustra quanto sia complicata la vita di un soggetto non udente, in famiglia, tra amici, a scuola, al lavoro ecc.

La soluzione proposta è un'app per smartphone che permettesse la trascrizione del parlato in tempo reale. Ogni interlocutore attiva il microfono e il bluetooth del proprio dispositivo e lo

---

<sup>26</sup> Campagna Indiegogo: <https://www.indiegogo.com/projects/ava-group-conversations-made-accessible#/>

smartphone della persona non udente si connette ai singoli dispositivi per ricevere la trascrizione del parlato.

Al momento il progetto rinominato AVA<sup>27</sup> appartenente al gruppo Transcense è in fase di testing.

## 3.2. Librerie per lo Speech-To-Text

Si illustrerà di seguito il procedimento in tre fasi utilizzato per la scelta del servizio di riconoscimento in tempo reale del parlato continuo come specificato nell'introduzione del primo capitolo.

### 3.2.1. Fase 1: Ricerca

In questa fase sono state ricercate le soluzioni proposte per il riconoscimento del parlato e in particolare lo speech-to-text. Verranno illustrate le principali caratteristiche di quanto è stato individuato per poter procedere, in base ad esse, alla fase di selezione. Seppur la finalità è quella di creare un servizio completo a costo zero, sono state incluse in questa fase anche le soluzioni commerciali per poter decidere se il costo di tali soluzioni fosse sopportabile. I servizi sono stati pertanto suddivisi in prima istanza tra servizi, librerie e API Free/Open-Source e servizi, librerie e API Commerciali.

#### Sphinx<sup>28</sup>

CMU Sphinx (in breve Sphinx) è il nome più generale riferito ad un gruppo di sistemi di Speech Recognition sviluppati nella Carnegie Mellon University (CMU)<sup>29</sup>.

Sphinx include una serie di riconoscitori vocali e un trainer di modelli acustici. Nel 2000 il gruppo Sphinx alla CMU si è dedicato allo sviluppo dei diversi componenti per i riconoscitori vocali open source. I decodificatori vocali sono completi di modelli acustici e applicazioni di esempio. Le risorse disponibili includono software addizionali per l'apprendimento (training) di modelli acustici, la compilazione degli stessi modelli e un dizionario di pronuncia pubblico (cmudict)<sup>30</sup>.

CMU Sphinx comprende le seguenti versioni:

---

<sup>27</sup> AVA è anche l'acronimo della patologia del sistema uditivo (Auditory Verbal Agnosia) meglio conosciuta come Sordità.

<sup>28</sup> Sito web di Sphinx: <http://cmusphinx.sourceforge.net/>

<sup>29</sup> Sito della CMU: <http://www.cmu.edu/>

<sup>30</sup> Dizionario 'cmudict': <http://www.speech.cs.cmu.edu/cgi-bin/cmudict>

**Sphinx**<sup>31</sup>: un sistema di riconoscimento vocale speaker-independent capace di riconoscere il parlato continuo, realizzato da Kai-Fu Lee<sup>32</sup>.

Sphinx è in realtà di solo interesse storico essendo pioniere del riconoscimento del parlato continuo indipendente dallo speaker; è stato sostituito in termini di prestazioni dalle versioni successive.

**Sphinx 2:** un riconoscitore dalle alte prestazioni, originariamente sviluppato da Xuedong Huang<sup>33</sup> presso la stessa CMU e rilasciato come open source su SourceForge nel 2000. Sphinx 2 si concentra sul riconoscimento in tempo reale per l'applicazione alla lingua parlata. Incorpora funzionalità avanzate come la generazione di ipotesi parziali, commutazione dinamica del modello linguistico ecc.

Il codice di Sphinx 2 è stato incorporato in diversi prodotti commerciali. Attualmente non è più in fase di sviluppo ma viene costantemente manutenuto.

**Sphinx 3:** adotta una rappresentazione molto più efficiente ed è utilizzato principalmente per il riconoscimento ad alta precisione, non in tempo reale. I recenti sviluppi (in algoritmi e in hardware) hanno reso Sphinx 3 “vicino” a prestazioni in tempo reale, anche se non ancora adatto per le applicazioni interattive critiche. Sphinx 3 è in fase di sviluppo attivo e in collaborazione con SphinxTrain (trainer del modello acustico) fornisce l'accesso a una serie di tecniche di modellazione che migliorano la precisione del riconoscimento .

**Sphinx 4:** è una completa riscrittura del motore Sphinx scritto interamente in linguaggio Java con l'obiettivo di fornire un quadro più flessibile per la ricerca nel riconoscimento vocale.

Julius<sup>34</sup>

Julius è un motore per il riconoscimento vocale open source dalle alte prestazioni che utilizza un ampio vocabolario per il riconoscimento del parlato continuo. Il motore è capace di riconoscere quasi in tempo reale la dettatura di 60.000 parole usando solo gli attuali comuni PC grazie agli algoritmi moderni più utilizzati incorporati nel motore.

Julius è stato sviluppato come parte di un toolkit gratuito per la ricerca, lo sviluppo è continuato Continuous Speech Recognition Consortium (CSRC) in Giappone dal 2000 al 2003.

---

<sup>31</sup> Sphinx (I vers.): [http://www.ri.cmu.edu/pub\\_files/pub2/lee\\_k\\_f\\_1990\\_1/lee\\_k\\_f\\_1990\\_1.pdf](http://www.ri.cmu.edu/pub_files/pub2/lee_k_f_1990_1/lee_k_f_1990_1.pdf)

<sup>32</sup> Kai-Fu Lee (December 3, 1961 Taipei, Taiwan) attualmente di nazionalità Americana, laureato alla Columbia University, dottore alla Carnegie Mellon University.

<sup>33</sup> Xuedong Huang (October 20, 1962, Hunan, China) attualmente di nazionalità americana, ha studiato presso la Hunan University, la Tsinghua University e la Edinburgh University; ha conseguito il dottorato e ha conseguito i premi Asian American Engineer of the Year 2011, IEEE 1993 Paper Awar, Allen Newell Research Excellence Medal.

<sup>34</sup> Sito web di Julius: [http://julius.osdn.jp/en\\_index.php](http://julius.osdn.jp/en_index.php)

Successivamente, a partire dalla rev.3.4, è stato integrato in Julius un parser riconoscitore basato sulla grammatica denominato "Julian". Julian permette la creazione a mano di grammatiche DFA (automi deterministici a stati finiti) come modello di linguaggio e permette di essere utilizzato per costruire sistemi di comando vocale con un piccolo vocabolario, o definire diversi task per un sistema di dialogo automatico.

Per eseguire il sistema di riconoscimento, è necessario un modello di linguaggio e un modello acustico per la propria lingua. Julius adotta modelli acustici in diversi formati standard.

Anche se Julius viene distribuito solo con i modelli giapponesi, il progetto VoxForge sta lavorando sulla creazione di modelli acustici anglofoni per l'uso del motore.

#### HTK<sup>35</sup>

Hidden Markov Model Toolkit (HTK) è un toolkit portatile per costruire e manipolare gli hidden Markov model.

HTK è usato principalmente per la ricerca nell'ambito del riconoscimento vocale, tuttavia è anche usato in numerose altre applicazioni come la sintesi vocale, riconoscimento dei caratteri e sequenziamento del DNA. HTK è composto da un set di librerie e tool scritti in linguaggio C. I tool forniscono soluzioni sofisticate per l'analisi del parlato, per il training dei modelli e per il test e analisi dei risultati.

HTK è stato originariamente sviluppato nel Machine Intelligence Laboratory<sup>36</sup> della Cambridge University, specificatamente all'Engineering Department (CUED) dove è stato utilizzato per costruire sistemi di riconoscimento vocale con un ampio vocabolario. Nel 1993, Entropic Research Laboratory Inc. ha acquisito i diritti per vendere HTK e il suo sviluppo è stato completamente trasferito in quei laboratori nel 1995 costituendo l'Entropic Cambridge Research Laboratory Ltd.

HTK è stato venduto a Microsoft 1999 assieme al laboratorio.

Attualmente il prodotto è distribuito open source con una particolare licenza che attribuisce a Microsoft il copyright del codice originale di HTK.

#### Kaldi<sup>37</sup>

Kaldi è un toolkit per il riconoscimento vocale disponibile con licenza Apache open source. Kaldi mira a fornire software flessibile ed estendibile, supporta diversi modelli di linguaggio.

---

<sup>35</sup> Sito web di HTK: <http://htk.eng.cam.ac.uk/>

<sup>36</sup> MIL (Machine Intelligence Laboratory) - University of Cambridge: <http://mi.eng.cam.ac.uk/>

<sup>37</sup> Articolo su Kaldi: [http://publications.idiap.ch/downloads/papers/2012/Povey\\_ASRU2011\\_2011.pdf](http://publications.idiap.ch/downloads/papers/2012/Povey_ASRU2011_2011.pdf)

Kaldi è nato nella Johns Hopkins University<sup>38</sup> nel 2009 in un workshop dal titolo "Low Development Cost, High Quality Speech Recognition for New Languages and Domains". L'obiettivo di questo progetto era un'indagine sull'apprendimento lessicale. Il software cominciò a essere sviluppato in tale contesto in maniera dipendente da HTK. Alcuni dei partecipanti del workshop hanno concordato di incontrarsi nell'estate del 2010 a Brno, Repubblica Ceca (ospitati dalla Brno University of Technology<sup>39</sup>). Lo scopo di questo seminario è stato quello di creare un prodotto basato sul lavoro svolto nel 2009 che fosse pulito e sganciabile, e per creare uno speech toolkit general purpose come sottoprodotto. Dopo tanto lavoro dedicato al prodotto, nel maggio 2011 è stato rilasciato il toolkit di base contenente un set funzionante di script. Attualmente il toolkit è ancora incompleto e si sta lavorando sodo per ottenere un prodotto finito.

### Java Speech API

Java Speech API (in breve JSAPI), è un set di API cross-platform che supporta il riconoscimento di comandi vocali, sistemi di dettatura e sintetizzatori vocali. Attualmente JSAPI include una documentazione delle API in javadoc per circa 70 classi e interfacce ma non fornisce il codice sorgente.

JSAPI fornisce le linee guida e le interfacce per creare un sistema completo per lo speech recognition; non fornisce tuttavia una implementazione di base o di esempio.

JSAPI è disponibile gratuitamente e gli sviluppatori invitano a contribuire allo sviluppo del prodotto. Una implementazione di JSAPI è IBM Speech for Java (divenuto IBM ViaVoice attualmente acquisito da Nuance<sup>40</sup>). Diverse altre implementazioni sono state avviate ma ormai chiuse o abbandonate. L'unica implementazione ancora in vita di JSAPI è JARVIS che sta deviando sull'utilizzo dei servizi Google per il riconoscimento del parlato.

---

<sup>38</sup> Johns Hopkins University <https://www.jhu.edu/>

<sup>39</sup> Brno University of Technology: <https://www.vutbr.cz/en/>

<sup>40</sup> Sito web Nuance: [http://shopit.nuance.com/store/nuanceeu/it\\_IT/DisplayHomePage](http://shopit.nuance.com/store/nuanceeu/it_IT/DisplayHomePage)

## RWTH ASR<sup>41</sup>

RWTH ASR (in breve RASR) è un toolkit di riconoscimento vocale proprietario che offre un ampio dizionario e il riconoscimento del parlato continuo.

Il toolkit include nuova tecnologie di riconoscimento vocale per lo sviluppo di sistemi automatici. Sviluppato dal Human Language Technology and Pattern Recognition Group della RWTH Aachen University<sup>42</sup>, RASR include strumenti per lo sviluppo di modelli acustici e decoder, nonché componenti per l'adattamento allo speaker, training del modello linguistico dello speaker e altri tipi di training.

Il toolkit è pubblicato sotto una licenza specifica "RWTH ASR" che ne garantisce l'uso gratuito, la ri-distribuzione e modifica per uso non commerciale.

## Google Web Speech API<sup>43</sup>

Con la versione numero 25 di Chrome, Google ha integrato il supporto per il riconoscimento vocale in molte lingue differenti tramite Web Speech API. Si tratta di una libreria JavaScript che permette agli sviluppatori di integrare facilmente feature di riconoscimento del parlato continuo per la dettatura nelle proprie web application. Le feature implementate in questa maniera possono essere utilizzate solo con il browser di casa Google, tuttavia, successivamente, Google ha rilasciato un set di API da utilizzare per applicazioni mobile su piattaforma Android.

## Mozilla Web Speech API<sup>44</sup>

Nasce come estensione del progetto GSoC<sup>45</sup> (Google Summer of Code), offre il supporto per comandi vocali nel browser Firefox e rappresenta un'estensione delle API disponibili nell'ambito dello Speech Recognition. I primi contributors al progetto sono Roshan Vidyashankar e Anant Narayanan. Le API sono strutturate in due componenti principali: una dedicata alla cattura dell'audio grezzo; l'altra per lo streaming dei dati audio al server di riconoscimento e per la ricezione del risultato in maniera asincrona.

---

<sup>41</sup> Pubblicazioni per RASR:

- O. Koller, H. Ney, and R. Bowden. Automatic Alignment of HamNoSys Subunits for Continuous Sign Language Recognition. In LREC Workshop on the Representation and Processing of Sign Languages: Corpus Mining, Portorož, Slovenia, May 2016.
- Z. Tüske, K. Irie, R. Schlüter, and H. Ney. Investigation on log-linear interpolation of multi-domain neural network language model. In IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pages 6005-6009, Shanghai, China, March 2016.

<sup>42</sup> RWTH Aachen University: <https://www.rwth-aachen.de/cms/~a/root/?lidx=1>

<sup>43</sup> Google Web Speech API: <https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html>

<sup>44</sup> Mozilla Web Speech API: [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Speech\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API)

<sup>45</sup> Sito GSoC: <https://summerofcode.withgoogle.com/organizations/6397769191260160/>

Al momento supporta diverse lingue ma è ancora in via di espansione, un mirror del progetto è disponibile su GitHub.

## VoxSigma

VoxSigma è un pacchetto di soluzioni software per l'elaborazione del parlato di ultima generazione creato da Vocapia Research.

Vocapia Research<sup>46</sup> è stata fondata nel 2000 come società di ricerca e sviluppo nel campo delle tecnologie orientate al parlato e al linguaggio. Vocapia ha una partnership con LIMSI<sup>47</sup>, un laboratorio CNRS francese che conduce ricerche sull'elaborazione del parlato dagli anni '70.

Vocapia Research propone soluzioni innovative per l'elaborazione e per il riconoscimento del parlato, per l'identificazione del linguaggio e il riconoscimento dello speaker in diverse lingue europee.

La raccolta di software VoxSigma è stato realizzato grazie alle più sofisticate tecniche sviluppate al LIMSI per la produzione del parlato e per la percezione dello stesso.

Alcuni dei più comuni ambiti in cui viene utilizzata questa tecnologia è l'estrazione di dati audiovisivi (broadcast, call center), monitoraggio e gestione dei media, sistemi conversazionali telefonici. VoxSigma è attualmente disponibile come suite software standalone e come Web service, inoltre fornisce delle API per Unix/Linux, C e C++ e la possibilità di esportare in formato XML.

La suite software VoxSigma offre un ampio vocabolario per lo speech-to-text in più lingue; include funzioni adattive che consentono la trascrizione del discorso in ambienti rumorosi, ad esempio con musica di sottofondo. La suite software è stata progettato per professionisti che hanno bisogno di trascrivere grandi quantità di documenti audio e video, sia in tempo reale, sia in modalità batch.

Il processo di conversione speech-to-text utilizzato è composto di tre fasi. Per prima cosa il software identifica i segmenti audio contenenti il parlato, successivamente identifica la lingua parlata, se non è noto a priori, e infine converte i segmenti individuati in testo. Il risultato dell'elaborazione è un documento XML completamente annotato che include le etichette per i segmenti contenenti del parlato e non, etichette che individuano gli speaker, le parole sono corredate di time code. Il tutto è realizzato con valori di confidenza di alta qualità. Il file XML ottenuto può essere direttamente indicizzato da un motore di ricerca, o in alternativa può essere convertito in testo normale con capitolazione e punteggiatura.

---

<sup>46</sup> Sito web Vocapia Research: <http://www.vocapia.com/>

<sup>47</sup> Sito web LIMSI: <https://www.limsi.fr/fr/>

## SpokenData<sup>48</sup>

SpokenData offre soluzioni commerciali e servizi cloud oriented per lo Speech to Text. Il servizio è di tipo asincrono e consente l'upload di file audio per ottenere la trascrizione degli stessi nel formato desiderato; inoltre offre la possibilità di acquistare il servizio di revisione da parte di un operatore.

Per gli sviluppatori offre delle REST API da integrare nelle applicazioni; le lingue supportate per il momento sono Inglese, Russo, Cinese, Spagnolo, Ceco e Slovacco.

## IBM Watson<sup>49</sup>

IBM Watson Developer Cloud (WDC), offre una varietà di servizi per sviluppare applicazioni cognitive. Ogni servizio di Watson fornisce una REST API per interagire con il servizio. Il servizio di Speech to Text incluso nella suite offre anche ulteriori interfacce.

Watson Speech to Text può essere usato ovunque fosse necessaria l'interazione vocale ed è consigliato per esperienze in mobilità. Le lingue attualmente supportate sono Inglese (UK/US), Giapponese, Spagnolo, Portoghese Brasiliano, Arabo moderno e Mandarino.

## LumenVox<sup>50</sup>

LumenVox Automated Speech Recognizer (ASR) è una soluzione software che converte audio parlato in testo, fornendo agli utenti un mezzo più efficiente di input. LumenVox ASR è integrato con più di 25 piattaforme vocali. Fornisce un'interfaccia API per sviluppatori o una soluzione basata su standard come Media Resource Control Protocol (CPRM). LumenVox Speech Recognizer supporta anche la comprensione del linguaggio naturale (NLU - Natural Language Understanding).

Le lingue supportate al momento sono Inglese, Spagnolo, Francese e Portoghese nei loro diversi dialetti.

LumenVox è realizzato in accordo con i principali standard citati nei capitoli precedenti per il riconoscimento vocale e la gestione delle grammatiche.

Le soluzioni proposte da LumenVox sono le seguenti:

Architettura client / server distribuita: Fornisce stabilità attraverso installazioni ridondanti e raggiunge i livelli più elevati di prestazioni attraverso il bilanciamento del carico, senza la necessità di una maggiore carico del processore.

---

<sup>48</sup> Sito SpokenData: <https://spokendata.com/>

<sup>49</sup> Sito IBM Watson: <http://www.ibm.com/watson/developercloud/speech-to-text.html>

<sup>50</sup> Sito LumenVox: <http://www.lumenvox.com/>

Licenza flessibile: permette di scegliere il modello di tariffazione giusta per il progetto con una selezione di opzioni di licenza che includono la tariffazione con abbonamento mensile, basata sull'uso e Software as a Service (SaaS).

Supporto agli Standard: l'adozione di protocolli standardizzati e strumenti di sviluppo rendono facile da sostituire l'ASR esistente con la soluzione LumenVox.

Grammatiche sul lato server: supporto efficiente per grammatiche di grandi dimensioni con l'utilizzo del caching.

Individuazione dell'attività vocale (Voice Activity Detection - VAD): separa con precisione parlato dal rumore di fondo in caso di utilizzo in ambienti molto rumorosi

N-Migliori Risultati: particolarmente efficace quando i chiamanti hanno bisogno di precisare i nomi, indirizzi stradali o indirizzi email. Invece di restituire solo il risultato con punteggio più alto, vengono restituiti alcuni dei risultati con i punteggi più alti, ossia le risposte più probabili. Il costo del servizio si aggira sul migliaio di dollari.

### SpeechMatics<sup>51</sup>

SpeechMatics prevede il riconoscimento basato su cloud del discorso basato sui più recenti progressi nelle reti neurali. La tecnologia è stata lanciata dal Dr Tony Robinson circa 30 anni fa e ora sta spingendo i confini di riconoscimento vocale automatico speaker-independent con alti livelli di precisione battendo in tutto il mondo. Il sistema basato su cloud offre una serie di vantaggi tra cui costi molto bassi di set-up, aggiornamenti automatici regolari e miglioramenti, nonché l'accesso a una gamma sempre crescente di strumenti e prodotti linguistici.

SpeechMatics è in grado di estrarre l'audio da quasi tutti i formati di file comuni e prevede anche il riconoscimento degli speaker, punteggiatura e maiuscole. I file di output possono essere in qualsiasi formato si richiede tra cui HTML, XML e PDF.

Il sistema è accessibile direttamente attraverso l'interfaccia web su server sicuro o tramite una semplice API REST.

### iSpeech<sup>52</sup>

iSpeech è un servizio per il riconoscimento vocale (ASR) e per la sintesi vocale (STT) che offre una semplice interfaccia di programmazione multi-piattaforma sia web che mobile.

iSpeech nasce nel 2011 ed è in continuo aggiornamento per offrire funzionalità sempre più ricche, al momento stanno lavorando per offrire il riconoscimento del parlato continuo in modalità contemporanea al parlato.

---

<sup>51</sup> Sito SpeechMatics: <https://www.speechmatics.com/about>

<sup>52</sup> Sito iSpeech: <https://www.ispeech.org/developers>

Al momento fornisce una SDK per la piattaforma Android, iOS e per l'integrazione con la maggior parte dei linguaggi di programmazione. Offre inoltre un ricco dizionario per il riconoscimento e un modello vocale per diverse lingue tra cui l'Italiano. Il prezzo del servizio è piuttosto esiguo e rappresenta un ottimo compromesso per realizzare prodotti di qualità. Diverse app sono già state sviluppate utilizzando iSpeech e sono disponibili sui principali mobile store.

### Api.ai<sup>53</sup>

Api.ai fornisce agli sviluppatori e alle aziende gli strumenti avanzati di cui hanno bisogno per costruire interfacce utente conversazionali per le applicazioni e i diversi dispositivi hardware. La piattaforma Api.ai consente agli sviluppatori di integrare perfettamente sistemi di comando vocale nei loro prodotti per creare interfacce utente più “friendly”.

Un prodotto sviluppato grazie ad Api.ai è Assistant<sup>54</sup>, prima app di assistente conversazionale creata nel 2010. Con oltre 20 milioni di utenti, è il più alto rating tra le app disponibili dello stesso genere assistente.

Api.ai fornisce un gran volume di API e SDK complete di documentazione dettagliata per le più svariate piattaforme e linguaggi, inoltre fornisce un ambiente di sviluppo online per preconfigurare il servizio necessario e integrarlo più facilmente all'interno del sistema su cui si sta lavorando.

Api.ai promette ottime prestazioni in diverse lingue tra cui Portoghese, Cinese, Inglese, Tedesco, Francese, Italiano, Giapponese, Russo e Spagnolo.

I prezzi per accedere al servizio vanno dal Free planning (per il testing fino a 6000 query al mese) fino al Premium (circa 1000\$ al mese per ottenere tutte le feature e il supporto disponibile).

### Bing Speech API

Bing Speech API è una API basata su cloud che offre algoritmi avanzati per elaborare il linguaggio parlato. Consente agli sviluppatori di aggiungere comandi vocali alle applicazioni, e l'interazione in tempo reale con l'utente.

Bing fornisce REST API cross-platform per consentire l'uso di funzionalità vocali su tutti i dispositivi collegati a Internet.

Microsoft ha utilizzato Bing Speech API per le applicazioni Windows come Cortana e Skype Translator così come le applicazioni Android come Bing, Android Wear e Android Phone.

---

<sup>53</sup> Sito Api.ai: <https://api.ai/>

<sup>54</sup> Assistant by Api.ai: <https://assistant.ai/>

L'API di riconoscimento vocale fornisce la capacità di convertire l'audio parlato in testo con l'invio di audio ai server di Microsoft nel cloud. Bing consente di scegliere se utilizzare le API REST o la libreria client.

Utilizzando l'API REST si otterrebbe un solo risultato finale senza risultati parziali. La documentazione per l'API REST è disponibile online completa di demo.

Utilizzando la libreria client è consentito lo streaming in tempo reale, il che significa che l'audio è inviato al server e i risultati del riconoscimento parziale vengono restituiti allo stesso tempo. Lo streaming in tempo reale è supportato su Android, iOS e Windows.

Bing Speech API supporta anche per convertire testo in audio che può riprodurre per l'utente. La conversione Text to Speech (TTS) è disponibile tramite API REST completa di documentazione e demo.

Le lingue supportate sono molteplici e comprendono l'Italiano.

I prezzi proposti sono molto accessibili, si va dal free testing fino a 5000 transazioni al mese, fino al pacchetto per il parlato continuo che comprende fino a oltre 100 ore al mese di parlato al prezzo di circa 6\$ l'ora.

### Nuance Dragon SDK<sup>55</sup>

Nuance è ormai diventata la casa simbolo del riconoscimento vocale ad alta qualità e ad alte prestazioni. Ha sempre investito su nuovi prodotti e tecnologie di Speech recognition e Text-to-Speech acquistando diversi brevetti provenienti da case note come IBM, Philips e molti altri sviluppatori indipendenti.

Al momento è sul mercato la potenza leader per il riconoscimento vocale speaker dependent e offre agli sviluppatori un SDK per poter implementare funzionalità di riconoscimento vocale sia su web con interfaccia REST API, sia in locale su diverse piattaforme. Le lingue e i dizionari supportati comprendono l'Italiano e molte altre lingue compresi i vari dialetti.

Il Dragon Software Developer Kit (SDK) è stato progettato per gli sviluppatori e gli integratori per aggiungere capacità avanzate di riconoscimento vocale in applicazioni in-house, applicazioni commerciali, ecc., utilizzando interfacce utente o workflow esistenti. Le soluzioni offerte sono di due tipi: Dragon SDK Client Edition (DSC) utilizzato per abilitare qualsiasi applicazione, incluse le applicazioni di reporting sottotitoli ecc.; Dragon SDK Server Edition (DSS) utilizzato per le applicazioni che richiedono il riconoscimento vocale accurato eseguito in background, come la trascrizione basata su server.

Nuance offre anche una piattaforma cloud di sviluppo (Mix) in beta che consente di supportare lo sviluppo su qualunque piattaforma.

---

<sup>55</sup> Nuance Developers: <https://developer.nuance.com/public/index.php?task=home>

I prezzi variano in base alle necessità e vanno dalla versione di test con le funzionalità di base gratuita fino a 20000 transazioni (silver plan); alla versione completa di supporto allo sviluppo con un costo di circa 25000\$ annuali (emerald plan).

### Microsoft Speech API<sup>56</sup>

Microsoft promette grazie all'interfaccia di programmazione di applicazioni SAPI (Speech API) di poter ridurre drasticamente il codice necessario per scrivere un'applicazione capace di utilizzare il riconoscimento vocale e la sintesi vocale, rendendo la tecnologia vocale più accessibile e affidabile per un'ampia gamma di applicazioni. Le API forniscono un'interfaccia ad alto livello tra le applicazioni e i motori di riconoscimento vocale. MSAPI implementa tutti i dettagli di basso livello necessari per controllare e gestire le operazioni in tempo reale dei vari motori di riconoscimento del discorso.

I due tipi fondamentali di motori SAPI sono sistemi text-to-speech (TTS) e di riconoscimento vocale (STT). I primi sintetizzano le stringhe di testo e file in audio parlato con voci sintetiche. I secondi convertono audio parlato in stringhe di testo leggibili da esseri umani.

Il programmatore può scegliere di utilizzare nella propria applicazione due diversi tipi di moduli di riconoscimento vocale. Un sistema di riconoscimento condiviso con altre applicazioni di riconoscimento vocale (raccomandato per la maggior parte delle applicazioni vocali) oppure, per grandi applicazioni server che devono essere in esecuzione autonomamente su un sistema, e per i quali ottenere alte prestazioni è la chiave di successo, un motore di riconoscimento vocale dedicato e più appropriato.

MSAPI fornisce anche delle grammatiche e vocabolari vasti in molteplici lingue tra cui l'Italiano. Con l'arrivo di Windows 10, il tutto è già integrato nel sistema operativo e il programmatore necessita solo di utilizzare chiamate tramite l'interfaccia API che dovrebbero consentire di utilizzare il riconoscimento vocale su tutti i sistemi Windows based. Il sistema così configurato è gratuito ma se si vogliono utilizzare ambienti di sviluppo cloud e servizi di riconoscimento cloud based per potervi accedere da qualsiasi piattaforma, ci si deve attenere ai piani di costo di tali sistemi facilmente integrabili con la nota piattaforma di casa Microsoft: Azure.

---

<sup>56</sup> MSDN - Microsoft Speech API: [https://msdn.microsoft.com/en-us/library/ee125663\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ee125663(v=vs.85).aspx)

I risultati individuati nella fase di ricerca sono riportati nella tabella riassuntiva seguente:

<b>Nome Servizio</b>	<b>Tipo</b>	<b>Licenza</b>	<b>Lingue</b>
<b>RWTH ASR</b>	Toolkit	GPL like	User Model
<b>Kaldi</b>	Toolkit	GPL like	User Model
<b>Julius</b>	Engine	GPL like	User Model
<b>SpokenData</b>	API	Commercial	Undefined
<b>Vocapia (VoxSigma)</b>	API	Commercial	Arabic, Dutch, English (US/UK), Finnish, French, German, Greek, Italian, Latvian, Lithuanian, Mandarin, Polish, Portuguese, Romanian, Russian, Spanish, Turkish
<b>Java Speech API</b>	API Spec.	GPL like	User Model
<b>Sphinx</b>	SDK	GPL like	User Model
<b>HTK</b>	Toolkit	GPL like	User Model
<b>IBM Watson</b>	API	Commercial	Brazilian Portuguese, Japanese, Mandarin Chinese, Modern Standard Arabic, Spanish, English (US/UK)
<b>LumenVox</b>	SDK	Commercial	English (US/UK/Australian/New Zealand), Spanish (Mexican/South American), Canadian French, Brazilian Portuguese
<b>SpeechMatics</b>	API	Commercial	Arabic, English (Australiian/British/North American)
<b>iSpeech</b>	API/SDK	Commercial	English (US/Canada/UK/Australia), Spanish (Spain/Mexico), Italian, French (France/Canada), Polish, Portuguese (Portugal/Brazil), Catalan, Chinese (Taiwan/China/Hong Kong), Danish, German, Finnish, Japanese, Korean, Dutch, Norwegian, Russian, Swedish
<b>Api.ai</b>	API/SDK	Commercial	Brazilian Portuguese, Chinese (Cantonese/Simplified/Traditional), English, Dutch, French, German, Italian, Japanese, Korean, Portuguese, Russian, Spanish, Ukrainian
<b>Mozilla Speech API</b>	API	GPL like	More then 70 Languages and Dialects
<b>Google Web Speech API</b>	API	GPL like	More then 70 Languages and Dialects
<b>Bing Speech API</b>	API/SDK	Commercial	More then 70 Languages and Dialects

<b>Nome Servizio</b>	<b>Tipo</b>	<b>Licenza</b>	<b>Lingue</b>
<b>Nuance Dragon SDK</b>	API/SDK	Commercial	Arabic (Egypt/Saudi Arabia/International), Bahasa (Indonesia), Cantonese (Simplified), Catalan, Croatian, Czech, Danish, Dutch, English (Australia/UK/US/India), Finnish, French(Canada/France), German, Greek, Hebrew, Hindi, Hungarian, Italian, Japanese, Korean, Malay, Mandarin (China/Simplified/Taiwan/Traditional), Norwegian, Polish, Portuguese (Brazil/Portugal), Romanian, Russian, Slova, Spanish, Swedish, Thai, Turkish, Ukrainian, Vietnamese
<b>MS Speech API</b>	API/SDK	GPL like	German, Chinese (Taiwan/Hong Kong/China), Russian, Spanish (Spain/Mexico), Japanese, Arabic, Danish, English (US/UK/Australia/Canada/Indonesian/New Zealand), Finnish, Dutch, Portuguese (Portugal/Brazil), Catalan, French (France/Canada), Korean, Swedish, Norwegian, Polish, Italian

### 3.2.2. Fase 2: Selezione

In questa fase, sono state individuate le caratteristiche vantaggiose e svantaggiose di ciascuna soluzione ed eliminate secondo specifici criteri quelle soluzioni che non soddisfano le necessità per questo progetto di tesi. In particolare, sono state scartate quelle soluzioni ancora non mature, in fase di sviluppo, in cui gli aggiornamenti sono poco frequenti e che non forniscono abbastanza strumenti per un prodotto affidabile. Per questi motivi sono stati esclusi Java Speech API, RWTH ASR, Sphinx Kaldi, Julius e HTK ancora privi di modelli vocali predefiniti che andrebbero quindi aggiunti in fase di sviluppo. La realizzazione di tali modelli porterebbe via troppo tempo per gli obiettivi che ci si pone in questo lavoro di tesi. Successivamente sono stati esclusi i servizi che supportano solo poche lingue, in particolar modo che non supportano l’Italiano (lingua su cui si svolgeranno la maggior parte dei test preliminari). Per questo motivo sono stati esclusi SpokenData, IBM Watson, LumenVox e SpeechMatics.

Da questa prima selezione, i candidati rimasti da analizzare sono quelli riportati nella tabella seguente. Per ciascuno di questi sono state effettuate ulteriori ricerche.

<b>Nome Servizio</b>	<b>Tipo di Riconoscimento</b>	<b>Costo Licenza</b>
<b>Vocapia (VoxSigma)</b>	Asynchronous	Not Defined
<b>iSpeech</b>	Asynchronous	Free - Mobile SDK; 0.0001\$ per word - REST API
<b>Api.ai</b>	Asynchronous	Free - 6.000 queries/month 89\$ Month (billed annually) - 160.000 queries/year
<b>Mozilla Speech API</b>	Asynchronous	Free
<b>Google Web Speech API</b>	Continuous (Streaming)	Free
<b>Bing Speech API</b>	Continuous (Streaming)	Free - 5000 transactions/month, 4\$ for every 1000 extra transactions
<b>Nuance Dragon SDK</b>	Asynchronous	Free - 20.000 transactions (SILVER); 0.008\$ per transactiona (GOLD); about 25.000\$ All inclusive (EMERALD)
<b>MS Speech API</b>	Asynchronous	Free

Dagli approfondimenti effettuati su costi e tipo di riconoscimento fornito, si è scelto di escludere dalla lista dei possibili servizi da utilizzare, VoxSigma, iSpeech, Mozilla Speech API, Api.ai e Microsoft Speech API poiché non supportano pienamente lo scopo per cui si sta realizzando il sistema, ovvero non supportano la trascrizione in streaming o real time (non permettono l'elaborazione del parlato e la restituzione del risultato mentre lo speaker sta ancora parlando).

A causa dei prezzi proibitivi, invece, è stato eliminato Nuance Dragon SDK che richiede la tariffa Emerald per fornire quello di cui si necessita.

Le restanti alternative saranno testate nella fase di adozione per scegliere quella che si adatta meglio alle esigenze.

### 3.2.3. Fase 3: Adozione

Nome Servizio	Pro	Contro
<b>Bing Speech API</b>	<ul style="list-style-type: none"> <li>• Numerose Lingue supportate</li> <li>• Offre risultati parziali (Real time)</li> <li>• Predice la punteggiatura</li> <li>• Nessun Vincolo (browser)</li> </ul>	<ul style="list-style-type: none"> <li>• Free - 5000 transactions/month , 4\$ for every 1000 extra transactions</li> <li>• Bassa qualità del riconoscimento</li> </ul>
<b>Google Web Speech API</b>	<ul style="list-style-type: none"> <li>• Numerose Lingue supportate</li> <li>• Alta qualità del riconoscimento</li> <li>• Offre risultati parziali (Real time)</li> <li>• Gratuito</li> </ul>	<ul style="list-style-type: none"> <li>• Vincolato a Chrome</li> <li>• Su smartphone da problemi di glitch</li> </ul>

Dopo un test preliminare effettuato grazie alle demo messe a disposizione dai fornitori di servizi, Bing Speech API è stato escluso a causa della scarsa qualità del riconoscimento. L'unica alternativa rimasta è quindi quella di casa Google nella sua versione web based vincolata al browser Chrome (Google Web Speech API) e nella versione Cloud-based legata per il deployment alla piattaforma cloud di Google (Google Cloud Speech API). La seconda soluzione prevede il pagamento dei servizi di cloud computing di Google che, seppur avendo prezzi esigui, si preferisce sfruttare nella versione “Free for try” ai fini dello sviluppo del lavoro di tesi.

In questa ultima fase, allo scopo di realizzare un core funzionante per analizzare la qualità del servizio, si è deciso di utilizzare solo il servizio Google Web Speech API che è completamente gratuito. Questo per evitare di consumare precocemente il periodo di prova offerto per l'utilizzo della piattaforma Google Cloud (legata a Google Cloud Speech API) e nella previsione di costruire un sistema abbastanza modulare da essere utilizzato con qualsivoglia servizio di riconoscimento del parlato si andrà a collegare.

## 4. Metodologia

Per lo sviluppo del sistema, si considera fondamentale poter controllare passo dopo passo come si evolve il sistema stesso, e che gli stakeholders accettino ogni incremento delle funzionalità. Per questo motivo, e per la necessità di portare a termine il lavoro in tempi piuttosto ridotti, si è deciso di adottare una metodologia di sviluppo agile che consentisse il controllo dell'avanzamento del sistema a livello qualitativo.

### 4.1. Metodologia di sviluppo Agile

A partire dai primi anni 2000, è emersa nel mondo dello sviluppo software, la necessità di superare alcuni limiti del classico modello di sviluppo a cascata ormai ben consolidato che andava però in contrasto con la rapida evoluzione delle tecnologie e di conseguenza bisogni degli stakeholders.

Il fulcro del processo di sviluppo software a cascata (“waterfall” in inglese) consiste nel ritenere che sistemi software complessi debbano essere realizzati in maniera sequenziale, per fasi necessariamente successive. Inizialmente vengono raccolti e analizzati tutti i requisiti; dopodiché viene eseguita la completa progettazione; infine viene implementato il sistema software, passando per una lunga fase di integrazione e test.

Questo approccio implica che i sistemi complessi debbano essere costruiti in un unico passaggio, senza rivisitare i requisiti o le idee di progettazione, alla luce di possibili mutamenti delle condizioni di business o tecnologiche. Tali rivisitazioni infatti, andrebbero ad impattare in maniera piuttosto gravosa sul lavoro svolto fino a quel momento rendendo vane molte delle scelte effettuate e molto del tempo investito.

Il processo a cascata è stato introdotto per la prima volta in un articolo<sup>57</sup> da Winston Royce nel 1970 e destinato ad essere utilizzato soprattutto in grandi progetti governativi.

Concettualmente il processo a cascata equivale a un nastro trasportatore in una linea di produzione: gli analisti dei requisiti compilano le specifiche di sistema da passare ai progettisti software che creano tutti i diagrammi necessari a documentare come il codice dovrà essere scritto. Gli schemi di progettazione sono poi passati agli sviluppatori, che implementano il codice così come è stato progettato (figura sotto).



<sup>57</sup> Royce W. "Managing the Development of Large Software Systems". Proc. Westcon, IEEE CS Press, 1970, pp. 328-339

Benché questi metodi sembrino funzionare in teoria, in pratica essi portano spesso al caos e a scarse prestazioni nel progetto. Analizzando difatti il processo a cascata più in dettaglio, possiamo comprendere quanto sia inevitabile che i tentativi di una specifica dei requisiti up-front (pianificata a priori) tralascerà alcuni dettagli molto importanti, semplicemente perché gli stakeholders non sono in grado di comunicare agli sviluppatori tutto ciò che serve sul sistema già all'inizio del progetto. Molti dei requisiti emergeranno infatti in modo chiaro solo successivamente.

Da circa una decina di anni è emersa una filosofia di sviluppo che è alla base di una vasta raccolta di nuovi processi nota come Agile Software Development. Tali nuovi processi si concentrano maggiormente sulle interazioni tra le persone e un rapido sviluppo di codice piuttosto che sulla documentazione e sulla pianificazione (che rimane però sempre presente e fondamentale per la buona qualità del processo e di conseguenza del prodotto).

Il significato letterale della parola agile è “caratterizzato da rapidità, leggerezza e facilità di movimento”. Questo significa quindi che il processo di sviluppo agile consiste in una consegna rapida di software caratterizzata da una maggiore facilità e flessibilità di sviluppo. Si potrebbe affermare che: “L'Agile è un processo di sviluppo che enfatizza la soddisfazione del cliente attraverso la fornitura continua di software funzionante”.

La formalizzazione dei principi su cui si basano le metodologie agili è stata oggetto del lavoro di un gruppo di guru dell'informatica che si sono riuniti in quella che hanno denominato “Agile Alliance”. Il documento nato da questo lavoro è stato sottoscritto da un nutrito gruppo di questi professionisti.

Tra i firmatari del Manifesto Agile<sup>58</sup> si possono individuare menti che hanno dato un notevole contributo allo sviluppo dei processi e prodotti informatici come li conosciamo oggi: Kent Beck, pioniere dei design pattern e dell'applicazione commerciale di Smalltalk, cocreatore del principale framework di unit testing per Java, JUnit; Ward Cunningham, fondatore del primo sito wiki: Portland Pattern Repository, il 25 marzo 1995, coautore del libro The Wiki Way; Ken Schwaber e Jeff Sutherland creatori del framework di sviluppo agile Scrum; e tanti altri.

L'obiettivo principale delle metodologie agili è la piena soddisfazione del cliente. Il corretto uso di queste metodologie, inoltre, può consentire di abbattere i costi e i tempi di sviluppo del software, aumentandone la qualità.

---

<sup>58</sup> Manifesto Agile: <http://www.agilemanifesto.org/iso/it/manifesto.html>

## **“Manifesto per lo Sviluppo Agile di Software**

*Stiamo scoprendo modi migliori di creare software,*

*sviluppandolo e aiutando gli altri a fare lo stesso.*

*Grazie a questa attività siamo arrivati a considerare importanti:*

***Gli individui e le interazioni*** più che i processi e gli strumenti

***Il software funzionante*** più che la documentazione esaustiva

***La collaborazione col cliente*** più che la negoziazione dei contratti

***Rispondere al cambiamento*** più che seguire un piano

*Ovvero, fermo restando il valore delle voci a destra,*

*consideriamo più importanti le voci a sinistra. “*

*Kent Beck  
Mike Beedle  
Arie van Bennekum  
Alistair Cockburn  
Ward Cunningham  
Martin Fowler*

*James Grenning  
Jim Highsmith  
Andrew Hunt  
Ron Jeffries  
Jon Kern  
Brian Marick*

*Robert C. Martin  
Steve Mellor  
Ken Schwaber  
Jeff Sutherland  
Dave Thomas*

I principi su cui si basa una metodologia agile che seguia i punti indicati dal Manifesto Agile, riportato sopra, sono solo quattro:

- le persone e le interazioni sono più importanti dei processi e degli strumenti (le relazioni e la comunicazione tra gli attori di un progetto software sono la miglior risorsa del progetto per evitare fraintendimenti e inevitabili fallimenti);
- è più importante avere software funzionante che documentazione (bisogna rilasciare nuove versioni del software ad intervalli frequenti, e bisogna mantenere il codice semplice e avanzato tecnicamente, riducendo la documentazione al minimo indispensabile);
- bisogna collaborare con i clienti oltre che rispettare il contratto (la collaborazione diretta offre risultati migliori dei rapporti contrattuali);
- bisogna essere pronti a rispondere ai cambiamenti oltre che aderire alla pianificazione (il team di sviluppo dovrebbe essere pronto, in ogni momento, a modificare le priorità di lavoro nel rispetto dell'obiettivo finale).

Potremmo dire che, indipendentemente dalla specifica metodologia seguita, il processo di sviluppo software agile è caratterizzato nel complesso dalle seguenti proprietà:

- **Iterativo e incrementale e con tempo fisso:** l'intera applicazione viene realizzata in unità incrementalì dette iterazioni. Il tempo di sviluppo di ciascuna iterazione è relativamente breve (mediamente due settimane), fisso e rigorosamente rispettato. Ogni iterazione rappresenta un mini incremento delle funzionalità del sistema che viene rilasciato come incremento del risultato dell'iterazione precedente.
- **Coinvolgimento attivo del cliente:** si ha un forte coinvolgimento del committente mediante collaborazione faccia-a-faccia. Il risultato di ogni iterazione viene testato e approvato dal cliente stesso. Il riscontro ottenuto (consigli, modifiche, criticità) è implementato in iterazioni successive, riducendo così i rischi e garantendo una maggiore soddisfazione del cliente.
- **Guidato dalle funzionalità con consegna basata su priorità:** viene posta una maggiore enfasi nel fornire le funzionalità più richieste nell'applicazione e maggiormente critiche. Alle funzionalità viene assegnata una priorità in base a esigenze del cliente, rischi di sviluppo, opportunità di business. Dapprima vengono sviluppate le funzionalità con priorità maggiore, quindi al termine di ogni iterazione, le priorità del progetto vengono rivalutate.
- **Adattiva:** la metodologia in generale è molto adattabile e flessibile, in modo tale che l'applicazione realizzata possa soddisfare l'afflusso di nuovi requisiti durante lo sviluppo. L'obiettivo è quello di adattarsi a necessità mutevoli.
- **Responsabilizzare il team:** i team di progetto sono in genere piccoli e caratterizzati da un alto grado di interazione e comunicazione. Dal momento che l'intero team è attivamente coinvolto, esso stesso ha il potere di prendere decisioni.
- **Centrato sulle persone:** viene posta una maggiore enfasi su come le persone più qualificate ad effettuare lo sviluppo lavorino assieme, che non sul seguire i processi. La documentazione e le altre attività non propriamente di sviluppo e test sono ridotte al minimo.
- **Sviluppo rapido:** in genere lo sviluppo viene effettuato velocemente, utilizzando moderne tecnologie di sviluppo leggere. Questo ha il vantaggio di consentire cicli di feedback ridotti.
- **Rilasci frequenti:** grazie al procedimento composto da piccoli passi, il team di sviluppo è in grado di produrre versioni del software in tempi più ridotti; i rilasci risultano quindi più frequenti.
- **Testing:** uno dei concetti cardine delle discipline agili è il testing, la verifica automatica del corretto funzionamento del sistema. Questo si applica sia al codice,

che ai dati, e agli altri prodotti a cui le diverse discipline sono orientate (come i modelli o la documentazione). Dove non è possibile un test automatico, viene proposta una verifica manuale da parte di colleghi di pari esperienza.

- **Più disciplina:** volendo essere veloci, tutto deve essere consegnato correttamente già la prima volta. Il processo implica molto gioco di squadra e autodisciplina.
- **Semplicità:** l'accento è posto sul mantenere ogni cosa il più semplice possibile ed essere aperti al cambiamento. È importante sottolineare che quest'ultimo aspetto è sempre presente nell'insieme dei valori di tutte le metodologie agili.
- **Controllo della versione:** una delle conseguenze dirette dell'iterazione nella produzione è la necessità di introdurre un modello, un metodo, uno strumento, per il controllo delle versioni del software prodotto e rilasciato.

## 4.2. La metodologia Scrum<sup>59</sup>

Scrum è un framework agile di sviluppo del software, iterativo ed incrementale creato e sviluppato da Ken Schwaber, Jeff Sutherland.

*"Scrum è un framework di processo utilizzato dai primi anni novanta per gestire lo sviluppo di prodotti complessi. Scrum non è un processo o una tecnica per costruire prodotti ma piuttosto è un framework all'interno del quale è possibile utilizzare vari processi e tecniche. Scrum rende chiara l'efficacia relativa del proprio product management e delle proprie pratiche di sviluppo così da poterle migliorare."*

*(Jeff Sutherland, La Guida a Scrum<sup>60</sup>)*

Il termine Scrum è mutuato dal termine del rugby che indica il pacchetto di mischia ed è evidentemente una metafora del team di sviluppo che deve lavorare insieme in modo che tutti gli attori del progetto spingano nella stessa direzione, agendo come un'unica entità coordinata.

Scrum si basa sulla teoria dei controlli empirici di analisi strumentale e funzionale di processo o empirismo, ovvero sui tre i pilastri che sostengono ogni implementazione del controllo empirico di processo:

---

<sup>59</sup> Il termine Scrum, in quanto applicato allo sviluppo di prodotti, è stato per la prima volta utilizzato in "New New Product Development Game" (Harvard Business Review 86116:137–146, 1986) e successivamente elaborato in "The Knowledge Creating Company" entrambi testi di Ikujiro Nonaka e Hirotaka Takeuchi (Oxford University Press, 1995).

<sup>60</sup> Ken Schwaber e Jeff Sutherland, *La Guida a Scrum. La Guida Definitiva a Scrum: Le Regole del Gioco*, Scrum.Org and ScrumInc, 2014.

- **Trasparenza:** gli aspetti significativi del processo devono essere visibili ai responsabili del lavoro. La trasparenza richiede che quegli aspetti siano definiti da uno standard comune in modo tale che gli osservatori condividano una comune comprensione di ciò che viene visto.
- **Ispezione:** chi utilizza Scrum deve ispezionare frequentemente gli artefatti prodotti ed i progressi realizzati verso il conseguimento degli obiettivi prestabiliti, individuando in tal modo precocemente eventuali difformità rispetto a quanto si intende realizzare. Le ispezioni dovrebbero essere eseguite diligentemente e da ispettori qualificati.
- **Adattamento:** se chi ispeziona verifica che uno o più aspetti del processo di produzione sono al di fuori dei limiti accettabili e che il prodotto finale non potrà essere accettato, deve intervenire sul processo stesso o sul materiale prodotto dalla lavorazione. L'intervento deve essere portato a termine il più rapidamente possibile per ridurre al minimo l'ulteriore scarto rispetto agli obiettivi prestabiliti. Scrum prescrive quattro occasioni formali per l'ispezione e l'adattamento:
  - Sprint Planning Meeting
  - Daily Scrum
  - Sprint Review
  - Sprint Retrospective

tutti adattabili alle esigenze del progetto in maniera, appunto, agile.

In maniera molto sintetica Scrum è un framework di processo che prevede di dividere il progetto in blocchi rapidi di lavoro (Sprint) alla fine di ciascuno dei quali creare un incremento del software. Esso indica come definire i dettagli del lavoro da fare nell'immediato futuro e prevede vari meeting con caratteristiche precise per creare occasioni di ispezione e controllo del lavoro svolto.

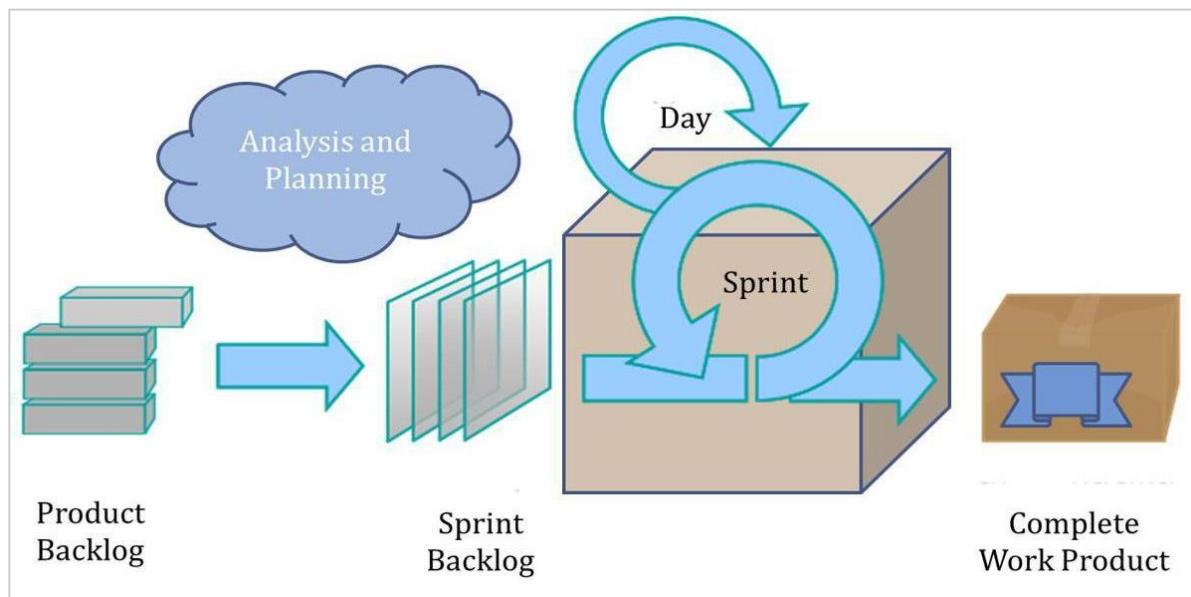
Le regole di Scrum legano insieme gli eventi, i ruoli e gli artefatti governando le relazioni e le interazioni tra essi anche se le strategie specifiche per l'utilizzo del framework Scrum variano in base alle esigenze.

I ruoli coinvolti per l'utilizzo efficace del framework sono individuati nelle seguenti figure che insieme costituiscono il Team Scrum e sono quelle persone impegnate nel progetto e che realizzano il prodotto (obiettivo del progetto).

- **Product Owner:** rappresenta gli stakeholders ed è la voce del cliente. Definisce i requisiti di prodotto centrati sui bisogni dei clienti (tipicamente user stories), assegna loro la priorità, e li aggiunge al product backlog. I team Scrum debbono avere un Product Owner e si raccomanda che questo ruolo non sia combinato con quello dello ScrumMaster.

- **Team di sviluppo:** è responsabile della consegna del prodotto, con incrementi di caratteristiche, che sia potenzialmente rilasciabile alla fine di ogni Sprint. Un Team di sviluppo è composto da 3-9 persone, con competenze differenti, che realizzano il lavoro effettivo.
- **Scrum Master:** è responsabile della rimozione degli ostacoli che limitano la capacità del team di raggiungere l'obiettivo dello Sprint e i deliverable previsti. Lo Scrum Master non è identificato come team leader, ma piuttosto colui che facilita una corretta esecuzione del processo. Lo Scrum Master detiene l'autorità relativa all'applicazione delle norme, spesso presiede le riunioni importanti e pone sfide alla squadra per migliorarla. Una parte fondamentale del ruolo di Scrum Master è quello di proteggere il Team di sviluppo e tenerlo concentrato sui compiti fungendo da cuscinetto verso qualsiasi influenza di distrazione.
- **Stakeholder:** sono per lo più clienti, venditori, le persone che permettono il progetto e per i quali il progetto produce i benefici concordati che ne giustificano la produzione. Sono coinvolti direttamente nel processo solo durante le Sprint Review.
- **Eventuali Manager:** controllano l'ambiente di lavoro.

In Scrum, uno Sprint è definito (come in figura sotto) come un insieme di task da svolgere organizzati in base alle priorità.



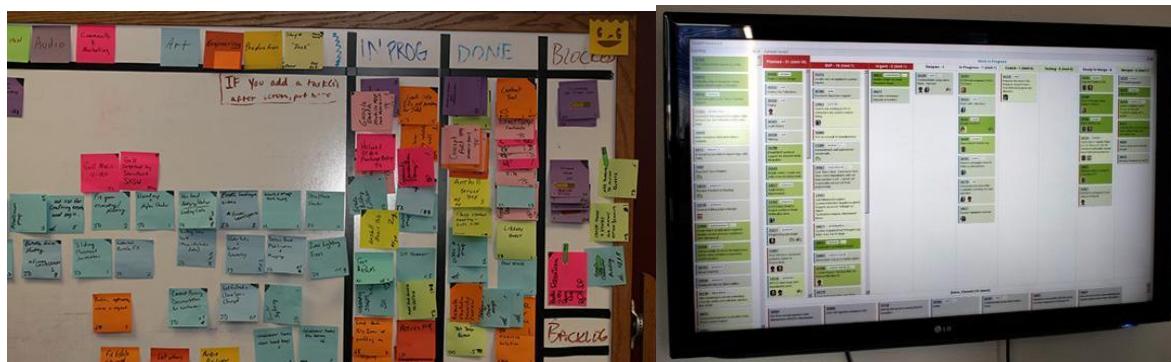
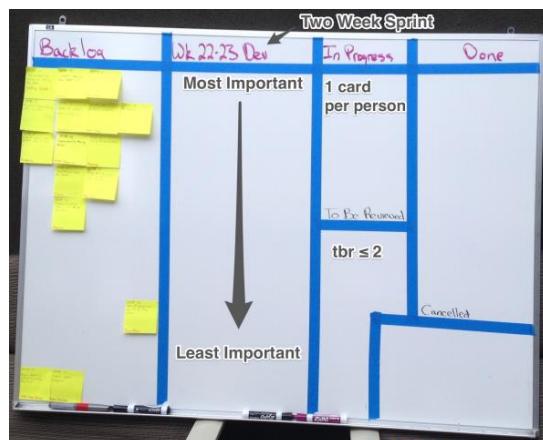
Lo sprint è un'unità di base dello sviluppo in Scrum e ha durata fissa, generalmente da una a quattro settimane.

Ogni Sprint è preceduto da una riunione di pianificazione in cui vengono identificati gli obiettivi e vengono stimati i tempi. Durante uno sprint non è permesso cambiare gli obiettivi, quindi le modifiche sono sospese fino alla successiva riunione di pianificazione, e potranno essere prese in considerazione nel successivo Sprint.

Al termine di ogni sprint il team di sviluppo consegna una versione potenzialmente completa e funzionante del prodotto, contenente gli avanzamenti decisi nella riunione di pianificazione dello sprint.

Nel corso di ogni sprint, il team crea porzioni complete di un prodotto. L'insieme delle funzionalità che vengono inserite in un determinato sprint provengono dal product "backlog", che è una lista ordinata (per priorità) di requisiti. La selezione di quali item del backlog verranno effettivamente inseriti nello sprint (a costituire l'obiettivo dello sprint o "sprint goal") viene effettuata durante lo sprint planning meeting. Se i requisiti non sono stati completati per una qualsiasi ragione vengono esclusi dalla review e reinseriti nel product backlog, o pianificati per lo sprint successivo.

Di notevole importanza per una buona applicazione di Scrum è il coordinamento e la comunicazione all'interno del team. allo scopo, si utilizzano quelle che vengono chiamate Scrum Task Boards, ovvero delle lavagne o intere pareti dedicate all'organizzazione dei task su classici post-it. Alternativamente possono essere utilizzate boards digitali. Nelle immagini successive è possibile individuare alcuni esempi.



## 4.3. Utilizzo di Scrum

Per quanto riguarda il progetto descritto in questo documento, la metodologia Scrum è stata applicata al fine di garantire un controllo continuo dell'avanzamento dei lavori e poter

affrontare facilmente i cambiamenti delle tecnologie utilizzate. Inoltre, per esigenze di tempo, la necessità di avere rilasci successivi è stata decisiva.

Il Team Scrum è composto da cinque elementi tra cui due Stakeholders (committenti), un Product Owner, uno sviluppatore e uno Scrum Master.

Prima dell'inizio dello sviluppo dei primi deliverable, sono stati predisposti alcuni sprint preliminari con durata maggiore rispetto ai successivi. In particolare i primi due sprint hanno avuto durata di circa due settimane mentre i successivi circa 9 giorni.

Durante il primo meeting, sono stati pianificati i task preliminari da svolgere per predisporre il progetto e dei requisiti fondamentali alla presenza di tutto il team.

Per il coordinamento dei task e per mantenere aggiornato in tempo reale l'avanzamento del lavoro, è stato predisposto un repository GitHub<sup>61</sup> a cui è stata associata una Scrum Board online: ZenHub.

#### 4.3.1. Controllo versione e GitHub

Il controllo versione non è altro che la gestione di versioni multiple di un insieme di informazioni quali file, programmi, documenti.

Gli strumenti per il controllo versione sono largamente utilizzati per progetti di sviluppo software.

Il controllo versione si è sviluppato negli anni a partire dai processi formali basati sui disegni cartacei. Le modifiche ai documenti sono identificate da un numero incrementale (codice) associato ad essi, denominato "numero di versione".

Nell'ingegneria del software, il controllo versione è qualunque pratica che tiene traccia e permette di controllare i cambiamenti al codice sorgente dei prodotti, per i file di documentazione e di configurazione. Il controllo di versione a livello di sviluppo software risulta fondamentale allo scopo di individuare e correggere i bug, è di importanza vitale per il programmatore poter recuperare e mandare in esecuzione diverse versioni del software per determinare in quali versioni il problema si è verificato o per la necessità di sviluppare parallelamente due versioni del software.

Nella maggior parte dei progetti di sviluppo software, più sviluppatori lavorano parallelamente sullo stesso software. Spesso capita che due sviluppatori tentano di modificare lo stesso file contemporaneamente. La maggior parte dei sistemi di controllo versione possono risolvere il problema dell'accesso concorrente in diversi modi.

I sistemi di controllo versione possono usare diversi modelli di archiviazione dei file versionati nel repository. Un modello è quello centralizzato, in cui tutte le funzioni di

---

<sup>61</sup> Repository del progetto: <https://github.com/collab-uniba/scriba.git>

controllo versione sono eseguite da un server condiviso. Da pochi anni alcuni sistemi hanno cominciato a usare un modello distribuito, in cui ogni sviluppatore lavora direttamente con il suo repository locale, e le modifiche sono condivise tra i repository in un passo separato. Altri sistemi invece, permettono a più sviluppatori di modificare lo stesso file nello stesso tempo, e forniscono degli strumenti per combinare le modifiche in seguito (merge).

Per riassumere, nei sistemi di controllo versione utilizzati in ambito di sviluppo software, il codice è memorizzato da qualche parte in un “repository” chiamato anche “depot”.

Ogni volta che vengono effettuate modifiche al codice, il responsabile effettua un “commit”; il software di controllo versione controlla quali file sono stati modificati dall'ultima sincronizzazione e rende disponibile a tutti gli altri sviluppatori le nuove modifiche. Tutte le modifiche apportate (change) rappresentano parte della storia del software ricostruibile nel tempo. In ogni commit, è possibile effettuare una serie di modifiche di cui si tiene traccia nella ”change list”.

I sistemi di controllo versione permettono inoltre di agire sull'intero repository attraverso azioni di ”check-out” ovvero viene effettuata una copia di lavoro dal repository oppure un ”update” (o sync) che copia le modifiche fatte sul repository nella propria directory di lavoro; ”merge” invece unisce modifiche concorrenti in una versione unificata

Tra i sistemi di controllo versione maggiormente utilizzati nel mondo dello sviluppo software OpenSource, si colloca Git nelle sue diverse varianti.

Git<sup>62</sup> nasce nel 2005 grazie a Linus Torvalds per essere un semplice strumento per facilitare lo sviluppo del kernel Linux, originariamente ospitato da BitKeeper che aveva rifiutato di mantenere il sistema per alcune controversie con i detentori del progetto.

Il progetto di Git deriva dall'esperienza di Torvalds nel mantenere un grande progetto di sviluppo distribuito e da un bisogno urgente di produrre un sistema soddisfacente in poco tempo. Queste necessità hanno condotto alle seguenti caratteristiche ereditate da Git:

- Forte supporto allo sviluppo non lineare: Git supporta diramazione e fusione (branching and merging) rapide e comode, e comprende strumenti specifici per visualizzare e navigare una cronologia di sviluppo non lineare.
- Sviluppo distribuito: Git dà a ogni sviluppatore una copia locale dell'intera cronologia di sviluppo, e le modifiche vengono copiate da un repository a un altro. Queste modifiche vengono importate come diramazioni aggiuntive di sviluppo, e possono essere fuse allo stesso modo di una diramazione sviluppata localmente.
- I repository possono essere pubblicati facilmente tramite i classici protocolli di rete più diffusi o uno speciale protocollo git.

---

<sup>62</sup> Git in lingua inglese significa “idiota”

- Gestione efficiente di grandi progetti: Git è molto veloce e scalabile, tipicamente un ordine di grandezza più veloce degli altri sistemi di controllo versione, e due ordini di grandezza più veloce per alcune operazioni.
- Autenticazione crittografica della cronologia: la cronologia di Git viene memorizzata in modo tale che il nome di una "commit" particolare dipende dalla completa cronologia di sviluppo che conduce a tale commit. Una volta che è stata pubblicata, non è più possibile cambiare le vecchie versioni senza che ciò venga notato.
- Portatile: Git è pensato per funzionare in tutti i sistemi operativi.

Da non confondere con Git, GitHub è un servizio di hosting per progetti software. Il nome deriva dal fatto che GitHub è un servizio sostitutivo del software dell'omonimo strumento di controllo versione distribuito, Git.

Il sito ha funzionalità simili ad un social network come feeds, follower e grafici per controllare come gli sviluppatori lavorano sulle varie versioni dei repository.

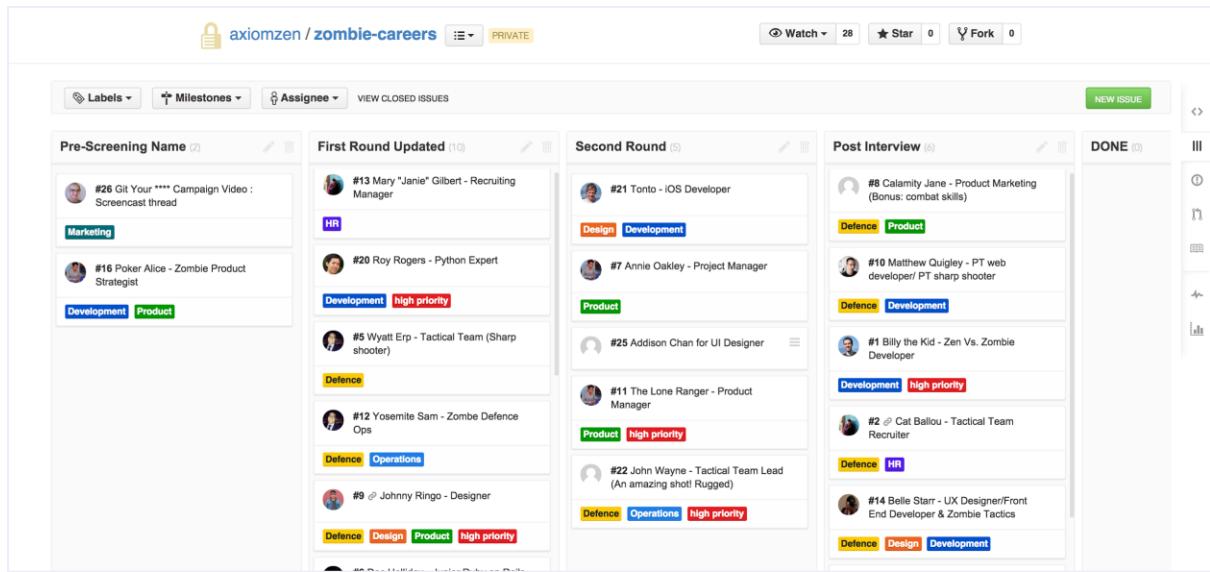
GitHub Inc. venne fondata nel 2008 con il nome di Logical Awesome dagli sviluppatori Chris Wanstrath PJ Hyett e Tom Preston-Werner. Nei primi mesi del 2009 raggiunse i 46.000 repository pubblici ospitati, mentre a metà dell'anno, i repository pubblici erano diventati 135.000, 1 milione l'anno successivo e 2 milioni nel 2011.

#### 4.3.2. ZenHub

ZenHub è una estensione per browser che migliora il flusso di lavoro di GitHub con caratteristiche costruite appositamente per i team di sviluppo agile. ZenHub offre boards interattive personalizzabili, possibilità di organizzare ed etichettare gli item, caricare file e documentazione, commenti e supporto alle Epic. ZenHub è completamente gratuito per i repository pubblici con team di cinque persone al massimo.

Il sistema è stato realizzato nel 2014 dalla Axiom Zen, una startup che ha fatto di ZenHub il suo core business.

ZenHub rappresenta quindi un incremento di produttività del sistema GitHub rendendo le sue funzionalità più semplici e rapide da usare e incrementandole con nuove feature caratteristiche della metodologia di sviluppo agile.



La caratteristica principale di questo plugin è proprio la board (figura sopra) in cui gli item o issue sono rappresentate come delle schede trascinabili tra le diverse colonne che stanno a rappresentare lo stato di avanzamento dell'Issue. Inoltre, ogni issue è etichettabile e assegnabile ad uno specifico sviluppatore o ad una specifica milestone o ancora ad una Epic: insieme di issue o task che sono concettualmente collegate a formare una issue più grande.

La gestione della board è completamente personalizzabile in base alle esigenze di progetto e si adatta perfettamente a molte delle metodologie di sviluppo più frequentemente utilizzate dagli sviluppatori cosiddetti “agili”.

## **5. Sprint 0 - Preliminare**

Durante il primo sprint o “Sprint 0”, si è rivelato necessario raccogliere a grandi linee i requisiti principali del sistema che si sta realizzando, in particolare è stato necessario capire quali tecnologie si adattano meglio alle necessità e come sfruttarle al massimo per ottenere un risultato prestante.

### **5.1. Analisi del contesto lavorativo**

Il contesto in cui il sistema proposto in questo documento di tesi dovrebbe lavorare è inquadrato in una qualsiasi aula adibita per l’occasione ad aula per conferenza, in cui saranno presenti uno o più relatori disposti frontalmente ad un pubblico seduto.

Il necessario perché il sistema possa lavorare è una connessione a Internet Wi-Fi o una connessione dati mobile (preferibilmente 4G/LTE per migliori prestazioni). Inoltre dovrebbe essere presente almeno un laptop/notebook per il collegamento al servizio via web.

Preferibilmente, ciascun relatore dovrà disporre di un dispositivo auricolare/microfono collegato in qualche modo al PC in modo da risultare periferica di input audio predefinita, in alternativa una periferica microfonica collegata alla macchina deve essere passato tra i relatori.

Nell’eventualità di proiezione di slide da parte dei relatori, l’aula dovrà inoltre disporre di una macchina collegata alla rete e ad un sistema di proiezione.

Gli ascoltatori interessati all’utilizzo del sistema dovranno disporre dell’accesso alla rete Wi-Fi o ad una propria rete mobile (preferibilmente 4G/LTE) dal proprio smartphone oltre che all’applicazione installata sullo stesso.

Il link/codice identificativo della stanza virtuale dovrà essere facilmente o raggiungibile, senza ostacolare lo svolgimento dell’evento, dagli utenti interessati.

### **5.2. Analisi degli utenti attesi**

Il sistema proposto è mirato all’inclusione di soggetti con deficit uditivi in eventi sociali quali seminari, conferenze, workshop e riunioni. Tuttavia può destare l’interesse di chiunque voglia seguire gli interventi leggendoli o rileggendoli in un secondo momento per coglierne determinati aspetti, senza vincoli alcuni.

In base a quanto appena affermato, il bacino di utenza che si intende raggiungere è composto da chiunque fosse interessato ad includere soggetti non udenti nel proprio evento come manager che volessero pubblicizzare un prodotto, organizzatori di eventi che volessero più

visibilità, relatori in eventi che volessero raggiungere un pubblico più ampio, docenti che volessero stimolare soggetti con determinati problemi a seguire le lezioni.

Dal punto di vista dei fruitori del servizio, invece, sicuramente si collocano tra gli utenti target quei soggetti che volessero assistere ad un evento pubblico personalmente e in totale indipendenza, soggetti (anche normodotati) che volessero tener traccia di quanto esposto durante l'evento pubblico, utenti semplicemente curiosi di provare il sistema proposto.

Roberto Polillo<sup>63</sup> in una delle sue pubblicazioni nella sua pubblicazione<sup>64</sup>, illustra come possono essere caratterizzate le diversità tra gli utenti di un sistema di qualsiasi genere. Esse si caratterizzano in livelli di una struttura piramidale in cui si distinguono i seguenti:

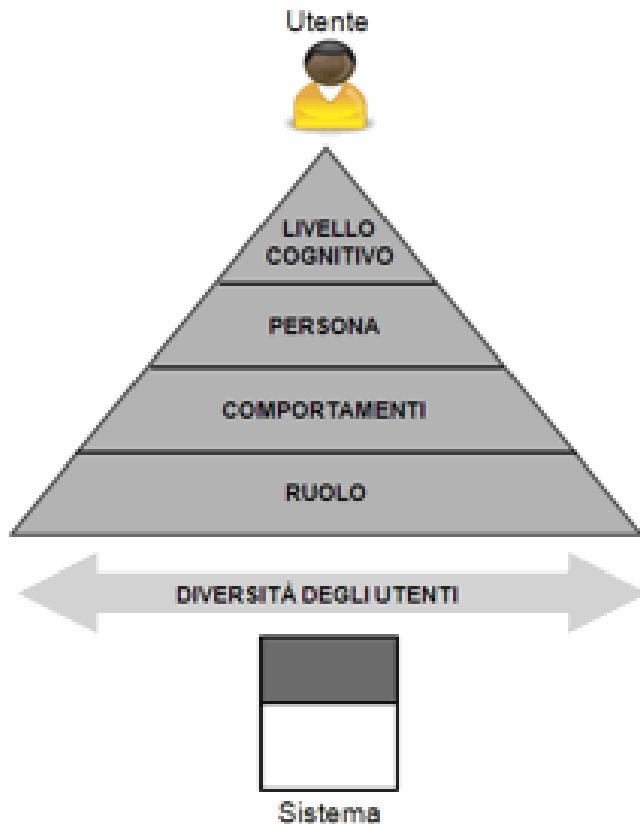
- **Livello Cognitivo:** si considera l'utente come essere umano dotato di specifiche abilità, derivanti dalla struttura del suo corpo e della sua mente. Una creatura che percepisce, interpreta, decide e agisce in un certo modo innanzitutto perché è dotato di un apparato sensoriale, cognitivo e motorio con specifiche caratteristiche, diverse da quelle di altre specie animali. Per fornirgli strumenti che possa utilizzare con profitto, è necessario conoscere queste caratteristiche: non è possibile chiedergli di svolgere compiti che non è fisicamente in grado di eseguire, e non è possibile pretendere elaborazioni che il suo cervello non può effettuare. A questo scopo, utilizziamo le conoscenze dell'ergonomia e dell'ergonomia cognitiva, che a loro volta si basano sulle conoscenze scientifiche derivanti dallo studio della psicologia e della fisiologia umana.
- **Livello Persona:** gli utenti vengono considerati ciascuno con una specifica formazione, cultura e storia personale che caratterizzano ogni specifico utente nella sua individualità. La persona specifica non sarà solo descritta da parametri che ne determinano le prestazioni fisiche e cognitive. Questa persona avrà un'età, un luogo di nascita, una lingua, una formazione, una professione, una storia personale. Avrà motivazioni, preferenze, interessi, amicizie, relazioni e sogni. Tutto ciò ne fa un individuo unico, diverso da ogni altra persona, e ne influenzera i comportamenti, in modo molto specifico.
- **Livello di Comportamento:** si devono considerare anche i rapporti che le persone hanno con gli altri, e in particolar modo con i membri delle comunità e organizzazioni cui appartengono. La conoscenza dei comportamenti degli utenti, nelle loro relazioni a uno specifico prodotto o servizio, ha un grande valore per il progettista. Solo conoscendone in dettaglio i comportamenti egli sarà in grado di comprenderne i problemi e le necessità, divenendo capace di individuare le soluzioni più adeguate.

---

63 Sito web R. Polillo: <http://www.rpolillo.it/>

64 R. Polillo - "Facile da usare - una moderna introduzione all'ingegneria all'ingegneria dell'usabilità", 2010

- **Livello di Ruolo:** un livello di analisi dell'utente ancora diverso è quello che ne analizza il suo ruolo in rapporto al sistema. Gli utenti di un sistema interattivo possono rivestire ruoli diversi, secondo la funzione che esercitano in rapporto ad esso. Per esempio, un utente di un sito web con il ruolo di amministratore svolge una funzione diversa da quella dei redattori dello stesso sito. Essi hanno compiti, responsabilità e diritti di accesso differenti.



*Roberto Polillo - "Facile da usare-Una moderna introduzione alla ingegneria dell'usabilità", Apogeo (2010), cap. 4*

Tenuto in considerazione quanto illustrato da Polillo, per il sistema proposto si possono effettuare le seguenti differenziazioni per ciascun livello:

- **Cognitivo:** Il sistema che si propone non richiede particolare sforzo cognitivo o ragionamento di alcun tipo, l'attenzione è posta sulla progettazione di un sistema adatto a tutti e completamente accessibile in modo da consentire a chiunque indiscriminatamente di accedervi, fatta eccezione per chi è affetto da disabilità visiva che non potrà farne uso considerata la natura stessa del sistema che si sta progettando orientato alla visualizzazione a schermo di testo.
- **Persona:** Il sistema è progettato per il momento per supportare la trascrizione di eventi in Italia, quindi in lingua italiana. Saranno per il momento esclusi soggetti che

non comprendono la lingua italiana. Non si esclude, in futuro, di inserire funzionalità di traduzione simultanea del testo in lingue diverse.

- **Comportamento:** A questo livello di analisi, il sistema ha lo scopo di uniformare il comportamento di soggetti non udenti a quello di soggetti sani in modo da consentire lo stesso tipo di interazione. Il sistema deve essere quanto più trasparente all'uso possibile in modo tale da fornire funzionalità ai non udenti (che molto probabilmente hanno difficoltà a parlare) per permettergli di porre domande al relatore come farebbe un uditore sano.
- **Ruolo:** Le differenze principali sugli utenti si individuano proprio a questo livello, i ruoli individuati nell'utilizzo del sistema sono tre:
  - **Creatore dell'evento:** che avrà la possibilità di gestire la stanza virtuale, i partecipanti virtuali ecc.
  - **Relatore:** che accederà come tale avendo la possibilità di trasmettere la propria voce al sistema che si occuperà di trascriverla. In alcuni contesti, il creatore dell'evento può ricoprire anche questo ruolo.
  - **Partecipante:** che potrà leggere ciò che viene proferito nella stanza virtuale dai relatori ed intervenire per porre i propri quesiti.

### 5.3. Analisi degli scenari

**Scenario 1:** Ada è una ragazza diciottenne appassionata di disegno artistico, ha frequentato il Liceo artistico con non poche difficoltà. Ada è infatti, come sua madre una ragazza non udente ed è riuscita a conseguire ottimi voti a scuola grazie all'aiuto dell'insegnante di sostegno che conosceva la LIS e l'ha aiutata a seguire le lezioni.

Ada ha pochi amici ma fedeli, i suoi amici hanno imparato a comunicare con Ada grazie al suo papà che ha insegnato loro la LIS.

Ada vorrebbe iscriversi all'Accademia delle Belle Arti e proseguire gli studi; sa bene che l'Università le metterebbe a disposizione un tutor che possa aiutarla durante le lezioni.

Ada comprende bene anche il labiale ma capita spesso che non riesca a trovare posto tra i primi banchi e non riesce a seguire i docenti quando sono voltati verso la lavagna. Specialmente quando il suo tutor non riesce ad accompagnarla, chiede aiuto ad un suo collega che usualmente registra le lezioni per poi sbobinarle tramite Facebook sullo smartphone poiché il suo collega non comprende la LIS, tuttavia Ada cerca sempre di evitare di disturbare il suo collega non essendo molto in intimità e non volendo sembrare un peso.

Ada ama stare tra i suoi amici più stretti ma non le dispiace affatto essere assieme a gente nuova, per questo frequenta molto volentieri l'università ma vorrebbe essere solo più indipendente.

**Scenario 2:** Luigi è un ragazzo di 30 anni laureato in Scienze della comunicazione, la sua tesi di Laurea era finalizzata a fornire una panoramica di come i media odierni e le nuove tecnologie abbiano cambiato il modo di comunicare e come sia cambiato l'approccio all'istruzione scolastica e accademica. Luigi è infatti appassionato di tecnologia e ama provare nuove tecnologie continuamente. Luigi era abituato a prendere appunti durante le lezioni, tuttavia aveva molte difficoltà a seguire il discorso dei docenti dovendo sintetizzare il tutto sul suo quaderno di appunti. Spesso ricorreva alla registrazione delle lezioni più importanti e alla successiva sbobinatura. Luigi ha provato anche diversi programmi per rendere il processo di sbobinatura automatico, erano però insoddisfacenti e poco affidabili a causa del rumore di sottofondo e al brusio dei suoi compagni registrati assieme alla lezione che non permettevano ai programmi di riconoscere la voce del docente. Per ovviare a questo problema, Luigi chiedeva al professore di posizionare il registratore sulla cattedra ma nel farlo si sentiva piuttosto in imbarazzo, quindi evitava spesso questa operazione. Oggi Luigi è disoccupato ma partecipa spesso a seminari e conferenze relative al suo ambito di studi. Anche in queste occasioni, Luigi si ritrova nella stessa situazione di quando seguiva le lezioni universitarie. Luigi vorrebbe tanto che quei programmini funzionassero correttamente perché gli piacerebbe avere una relazione scritta automaticamente di tutto ciò che veniva detto ai seminari a cui partecipava, poiché da esse avrebbe tratto spunto per nuove idee e progetti.

**Scenario 3:** Sonia è una psicologa del lavoro, ha 40 anni e insegna all'Università al dipartimento di Psicologia. Sonia ha una vera e propria passione per la psicologia e ama ascoltare la gente e discutere sugli argomenti più vari. Essendo una brava oratrice, Sonia è anche una brava scrittrice, le piace raccontare, scrivere piccoli romanzi senza grosse pretese. A Sonia capita spesso di presentare i suoi libri ad un pubblico nei centri culturali, nelle librerie e ha molto piacere di alternare questo tipo di eventi alle sue solite lezioni e seminari di psicologia.

A Sonia piace parlare tanto e spesso i ragazzi che seguono le sue lezioni non riescono a stare al passo, alcuni smettono di prendere appunti, altri ignorano la lezione affidandosi alle registrazioni di qualche compagno. Questo dispiace tanto a Sonia che non ama scrivere slide e dispense per i suoi studenti poiché ama spaziare tra i contesti durante le sue spiegazioni. Sonia vorrebbe che i suoi studenti potessero vivere la propria esperienza come fosse unica e le piacerebbe registrare le sue stesse lezioni per poterle scrivere il libro delle lezioni

dell'anno, ma ovviamente questo non è possibile perché i ragazzi pretendono di avere il materiale da studiare prima del termine del corso.

Nell'ambito delle varie presentazioni dei suoi libri, a Sonia è capitato spesso di conoscere persone non udenti che si rifugiano tra le righe dei suoi libri per passare un po' di tempo in tranquillità. A Sonia piacerebbe tanto se anche queste persone potessero ascoltare le sue parole e le sue riflessioni a proposito dei suoi scritti.

## 5.4. User Story

La metodologia Agile comprende molte pratiche da attuare durante il ciclo di vita del progetto, il punto di partenza è la scrittura delle user stories. Una user story può essere definita come la descrizione ad alto livello di una funzionalità utile al raggiungimento di un obiettivo di business. Con ‘storia’ viene definita in maniera non dettagliata una funzionalità che un software deve avere e che dà valore al prodotto finale consegnato al cliente/utente.

Una ‘storia’ permette di descrivere ed evidenziare l’importanza e l’impatto che una funzionalità avrà nel business, aiuta a far capire e decidere al cliente l’utilità della funzione stessa e la sua priorità, in alcuni casi fino a decidere di non realizzarla affatto. Le user stories evitano di scrivere verbose specifiche up-front che diventerebbero obsolete poco dopo averle scritte dato che, nel mondo reale, le cose cambiano molto più velocemente di quello che si immagina.

Il formato classico di una storia è il seguente:

COME < ruolo >

VOGLIO < fare qualcosa >

COSÌ CHE < possa ottenere valore per il business >

Una user story dovrebbero essere **INVEST**:

- **Independent:** ogni storia deve essere indipendente dall’altra. Anche se è molto difficile, si dovrebbero evitare situazioni nelle quali per fare una storia dobbiamo per forza farne un’altra prima. Avere storie indipendenti rende più liberi di spostare la priorità a piacimento.
- **Negotiable:** una storia non definisce contratti o requisiti che il software deve implementare. La storia definisce una breve e poco dettagliata descrizione della funzionalità che dovrà essere discussa e probabilmente modificata, riscritta o eliminata.
- **Valuable:** misurabile in termini di valore da tutti gli attori coinvolti. Ogni storia deve consegnare valore al suo utilizzatore finale (tipicamente utente/cliente).

- **Estimable:** si dovrebbe essere sempre in grado di stimare le storie che si scrivono.
- **Small:** le storie devono essere ragionevolmente piccole ovvero non devono mettere in difficoltà in fase di planning e prioritizzazione. Scrivere storie troppo grandi aumenta la probabilità di sbagliare la stima e di allungare i tempi di feedback. Storie grandi (dette epic) vanno spezzate in storie più piccole.
- **Testable:** le storie devono essere scritte in modo tale da poter essere testate. Quando una storia passa i suoi test significa che molto probabilmente è stata sviluppata correttamente. Non possono esistere storie che non possano essere testate.

Le storie estratte da un'analisi delle necessità che il sistema dovrebbe soddisfare sono le seguenti:

- Come **organizzatore** di eventi vorrei programmare un evento privato in tutte le sue parti in modo che sia memorizzato e accessibile da altri utenti autorizzati.
- Come **organizzatore** di eventi vorrei programmare un evento pubblico in tutte le sue parti in modo che sia memorizzato e accessibile da altri utenti.
- Come **organizzatore** di eventi vorrei gestire gli eventi programmati e quelli passati.
  - Per “gestire” si intende:
    - Creare un evento
    - Modificare un evento in ciascuna delle sue componenti
    - Eliminare un evento
- Come **organizzatore** di eventi vorrei invitare i relatori via e-mail in modo da coinvolgerli nell'evento.
- Come **organizzatore** di eventi vorrei invitare i partecipanti via e-mail in modo che siano a conoscenza dell'evento.
- Come **organizzatore** di eventi vorrei invitare i partecipanti via chat in modo che siano a conoscenza dell'evento.
- Come **organizzatore** di eventi vorrei gestire i partecipanti.
  - Per “gestire” si intende:
    - Accettare un partecipante
    - Rifiutare un partecipante
    - Eliminare un partecipante
- Come **organizzatore** di eventi vorrei divulgare un evento pubblico via social in modo che nuovi utenti siano a conoscenza dell'evento.
- Come **relatore** vorrei registrarmi ad un evento a cui sono stato invitato in modo da accettare un invito.
- Come **relatore** vorrei avviare e fermare i miei interventi in modo che le mie parole possano essere trascritte.

- Come **relatore** vorrei gestire gli eventi programmati a cui parteciperò.
  - Per “gestire” si intende:
    - Creare intervento
    - Eliminare intervento
    - Modificare intervento
- Come **relatore** vorrei gestire gli eventi passati a cui ho partecipato (passati).
  - Per “gestire” si intende:
    - Modificare testo intervento
    - Eliminare un evento
- Come **partecipante** vorrei registrarmi ad un evento pubblico in modo da potervi accedere e seguire la trascrizione.
- Come **partecipante** vorrei registrarmi ad un evento privato a cui sono stato invitato in modo da potervi accedere e seguire la trascrizione.
- Come **partecipante** vorrei fare un intervento (domanda) in modo che lo speaker la riceva e mi risponda.
- Come **partecipante** vorrei poter eliminare la partecipazione agli eventi programmati a cui ho accettato di partecipare in modo da ripulire la mia bacheca di eventi.
- Come **partecipante** vorrei gestire gli eventi passati a cui ho partecipato.
  - Per “gestire” si intende:
    - Eliminare l’evento dai miei eventi
    - Salvare l’evento in locale
    - Rileggere l’evento
- Come **utente** vorrei cercare un evento pubblico programmato o passato in base ai seguenti filtri:
  - Per Titolo
  - Per Data
  - Per Organizzatore
  - Per Relatore
- Come **utente** vorrei cercare un evento privato programmato o passato a cui sono stato invitato in base ai seguenti filtri:
  - Per Titolo
  - Per Data
  - Per Organizzatore
  - Per Relatore

- Come **utente** vorrei condividere un evento pubblico con i miei contatti via e-mail in modo che ne vengano a conoscenza.
- Come **utente** vorrei condividere un evento pubblico con i miei contatti via social in modo che altri ne vengano a conoscenza.

Per Evento si intende un oggetto così strutturato:

- **Evento:**
  - Titolo: Titolo dell'evento
  - Data Inizio: Giorno e ora in cui inizierà l'evento
  - Data Fine: Giorno e ora in cui terminerà l'evento
  - Permessi: L'evento può essere Privato o Pubblico, nel primo caso può accedervi solo chi ha un invito, nel secondo caso, può accedervi chiunque
- **Sessioni:**
  - Titolo: Titolo della singola sessione dell'evento
  - Data Inizio: Giorno e ora in cui inizierà la sessione
  - Data Fine: Giorno e ora in cui terminerà la sessione
- **Interventi:**
  - Titolo: Titolo del singolo intervento
  - Data Inizio: Giorno e ora in cui inizierà l'intervento
  - Durata: Durata stimata dell'intervento
  - Relatore: Chi parlerà durante l'intervento

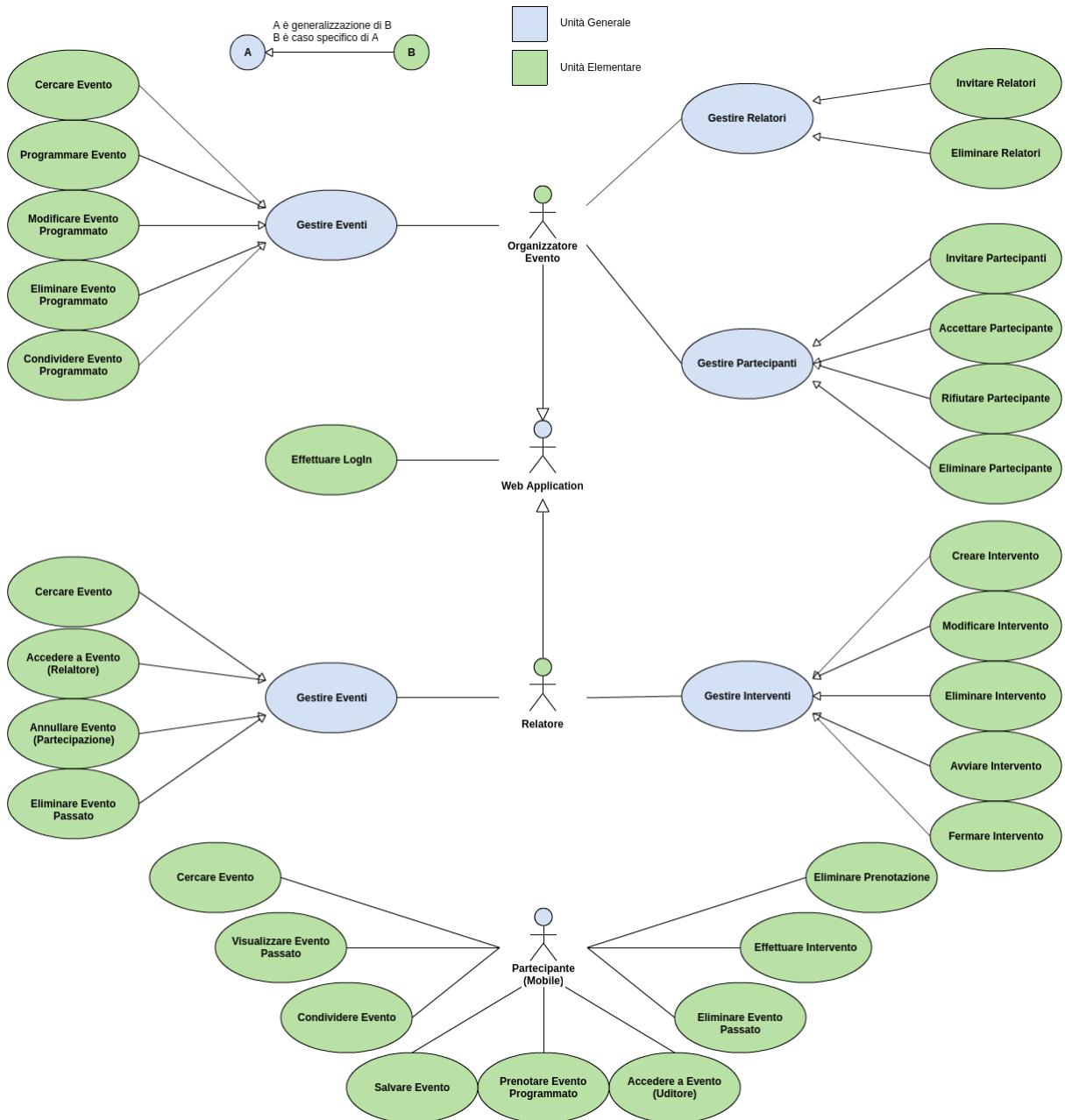
## 5.5. Casi d'uso

A partire dall'elicitazione dei requisiti principali come User Story, sono illustrati schematicamente nella figura seguente i casi d'uso principali associati ad ogni utente (ruolo). Ad ogni sprint successivo sarà pianificata la realizzazione (codifica) di alcuni di essi saranno predisposti i casi di test che verranno utilizzati per simulare l'interazione dell'utente e verificare il corretto funzionamento del sistema.

I casi di test individuati per ogni User Story o Item, saranno analizzati nel dettaglio man mano che vengono pianificati per lo sviluppo. In particolare saranno presi in considerazione i seguenti aspetti:

- **Caso d'uso:** indica il nome attribuito al caso d'uso specifico;
- **ID:** identificatore univoco del caso d'uso;
- **Descrizione:** una breve spiegazione del caso d'uso;

- **Pre-condizioni:** condizioni che devono necessariamente essere soddisfatte affinché il caso d'uso possa essere avviato;
- **Post-condizioni per Successo:** condizioni che saranno verificate qualora il caso d'uso terminasse con successo (fallimento del test: nessun errore);
- **Post-condizioni per Fallimento:** condizioni che saranno verificate qualora il caso d'uso terminasse con un fallimento (successo del test: errore);
- **Evento innescante:** azione che compie l'attore per attivare il caso d'uso specifico;
- **Attori primari:** rappresenta la persona fisica o un altro sistema che può attivare il caso d'uso.



Il passo successivo all'analisi dei requisiti principali, è stato quello di creare un repository per ospitare il sistema durante lo sviluppo e mantenere il controllo sulle versioni man mano che vengono apportate modifiche e miglioramenti. La scelta è ricaduta come preaccennato su GitHub che supporta gratuitamente la gestione del controllo versione di progetti con meno di cinque membri nel team, specialmente se open source. Il progetto che si sta realizzando è finalizzato a scopi sociali quindi il contributo di eventuali altri sviluppatori sarebbe apprezzato qualora ne avessero interesse. Tuttavia, i permessi di accesso al codice potranno essere modificati in un secondo momento se dovesse risultare opportuno.

Dopo la creazione e l'inizializzazione del repository, è stato installato il plug-in ZenHub per poter gestire meglio le issue e la collaborazione tra i membri del team. ZenHub è stato configurato per avere le seguenti pipeline:

- **Product Backlog:** vengono inserite tutte le issue proposte (user story e task) il cui sviluppo sarà programmato durante gli sprint.
- **Sprint Backlog:** vengono inseriti tutti i task programmati per il successivo sprint, scelti tra quelli più urgenti presenti nella pipeline precedente. Durante i meeting le issue da inserire per lo sprint successivo vengono accettati da tutto il team prima di procedere allo sviluppo.
- **In Progress:** vengono inserite una per volta (per ogni sviluppatore) le issue su cui si sta lavorando.
- **Review/QA:** vengono inserite tutte le issue completate che verranno testate e discusse durante il primo meeting programmato inerente allo sprint in corso.
- **Done:** vengono inserite le issue che hanno superato il test e che vengono quindi accettate dall'intero team. Gli sviluppi che passano in "done" sono pronti al rilascio.
- **Closed:** vengono inserite tutte quelle issue che con molta probabilità non saranno più utilizzate. Rappresenta una specie di recicle bin da cui potranno essere ripescate funzionalità che in un momento successivo risulteranno più fattibili o interessanti.

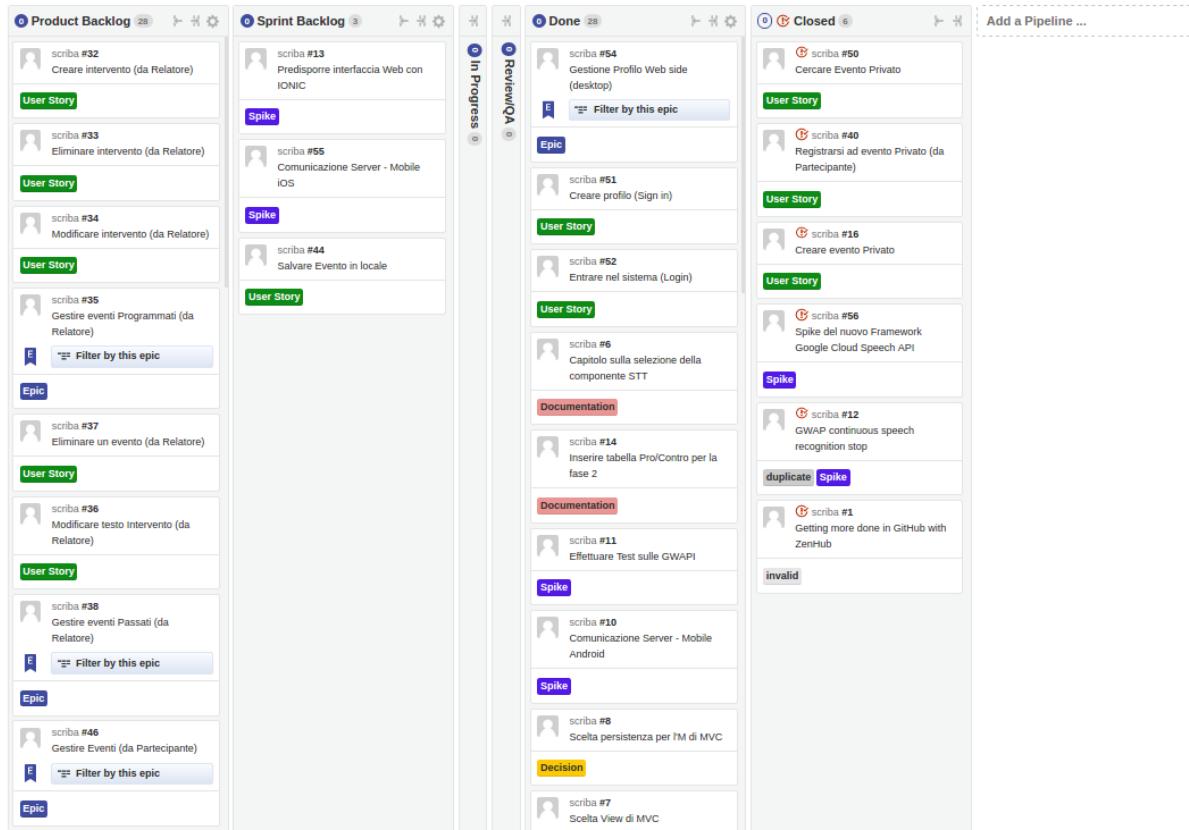
Ogni Issue è caratterizzata da un titolo semplice e diretto e una descrizione. Inoltre ad ogni issue è associata una label di colore differente tra le seguenti:

- **User Story (Verde):** sono gli avanzamenti effettivi nelle funzionalità del sistema, nuove funzionalità che soddisfano precise esigenze dell'utente. Questo tipo di issues derivano direttamente dai requisiti precedentemente elicitati.
- **Spike (Viola):** rappresentano dei test di fattibilità di determinate scelte, delle prove che ciò che si sta tentando di fare possa funzionare in maniera accettabile.
- **Decision (Rosa):** rappresentano momenti di scelta all'interno del progetto, sono utilizzate quando si deve decidere quale tecnologia utilizzare o quale strada

intraprendere. Sono documentate in modo da poter tornare sulla scelta in un secondo momento.

- **Documentation (Giallo)**: rappresentano momenti di scrittura del documento finale e di documentazione del codice, finalizzate a mantenere traccia dei task svolti al di fuori della programmazione, ai fini di questo lavoro di tesi.
- **Enhancement (Celeste)**: rappresentano piccoli miglioramenti del sistema da affrontare con priorità più bassa rispetto alle altre issue.
- **Epic (Blu)**: sono particolari issue che per la loro complessità non possono essere completate interamente in un singolo sprint e che possano essere scomposte in issue a granularità più fine da completare in sprint successivi. Sono rappresentate come Epic le gestioni (CRUD) di oggetti e dei profili utente. La particolarità delle Epic è che raggruppano concettualmente diverse issue e che vengono spostate nella pipeline successiva solo dopo aver spostato l'ultima issue a granularità più fine ad esse inerenti.
- **Duplicate (Grigio scuro)**: rappresentano issue inserite per errore più volte o con titoli diversi che si riferiscono allo stesso task. Queste issue vanno spostate nella pipeline “Closed” previa motivazione.
- **Invalid (Grigio chiaro)**: rappresentano issue inserite per errore o non più inerenti al sistema. Queste issue vanno spostate nella pipeline “Closed” previa motivazione.

Come fase culminante di questo sprint preliminare, sono state create ed inserite nella pipeline “Product Backlog” tutte le issue ritenute necessarie per lo sviluppo del sistema. Importante è sottolineare che ogni membro del team può inserire nuove issue (o modificarne) in qualsiasi momento (di seguito un’immagine di come appare la bacheca).



## 6. Sprint 1 - Scelta delle Tecnologie

Durante il meeting finale dello sprint preliminare, è stato deciso da tutti i membri del team di iniziare lo sviluppo scegliendo le tecnologie di base che avrebbero dovuto formare lo scheletro del sistema per poi poter espanderne le funzionalità aggiungendo nuove tecnologie e plugin. In particolare è stata programmata una fase di decisione delle tecnologie lato server, lato persistenza (database) e lato client (visualizzazione e controllo). Di seguito verranno illustrate le tecnologie selezionate con le loro principali feature.

### 5.1. Lato Server: NodeJS<sup>65</sup> (6.3.1)

NodeJS vede nascere la sua prima versione il 27 maggio 2009 con licenza MIT (appartenente al gruppo delle licenze libere GPL-like). NodeJS (o anche semplicemente Node) è una piattaforma event-driven studiata per il motore JavaScript V8, un motore JavaScript open source sviluppato da Google, attualmente incluso in Google Chrome, scritto in C++ e che supporta ECMAScript così come specificato nello standard ECMA-262, terza edizione<sup>66</sup>. V8 può essere incorporato in un software (come nel caso di Google Chrome) o può essere eseguito stand-alone, ad esempio in server destinati a supportare il framework Node. Sebbene Node non sia un framework JavaScript, molti dei suoi moduli base sono scritti in JavaScript, e gli sviluppatori possono scrivere nuovi moduli in JavaScript.

Il modello di networking su cui si basa Node non è quello dei processi concorrenti, ma I/O event-driven: ciò vuol dire che Node richiede al sistema operativo di ricevere notifiche al verificarsi di determinati eventi, e rimane quindi in attesa fino alla notifica stessa: solo in tale momento si attiva per eseguire le istruzioni previste nella funzione chiamante. Tale modello di networking è ritenuto più efficiente nelle situazioni critiche in cui si verifica un elevato traffico di rete<sup>67</sup>.

La normale procedura di esecuzione di una applicazione server-side, prevede una serie di istruzioni che effettuano frequentemente chiamate a servizi esterni come query per un database, o accesso al filesystem per operazioni di lettura e scrittura. Tutti questi eventi sono bloccanti, ovvero l'esecuzione dello script viene fermata finché non riceve una risposta dal componente a cui l'ha richiesta.

---

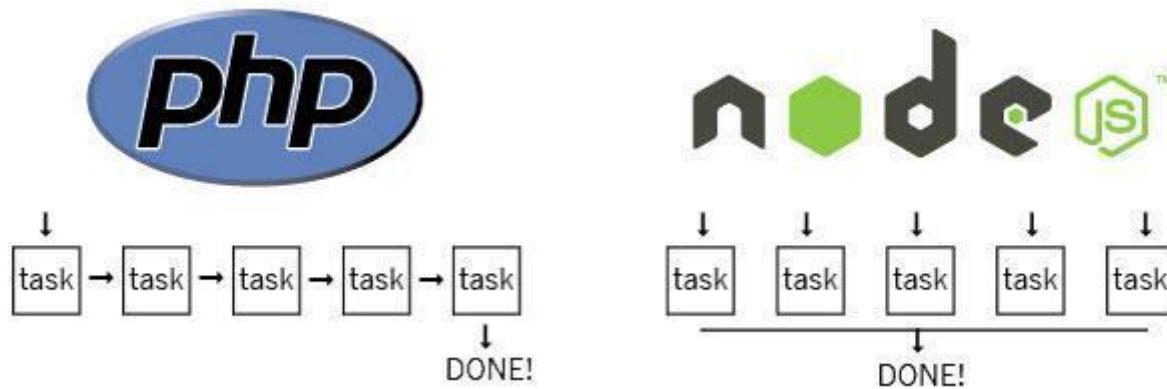
<sup>65</sup> Node.js: <https://nodejs.org/en/>

<sup>66</sup> Progetto V8: <https://bugs.chromium.org/p/v8/issues/list>

<sup>67</sup> Gestione server tramite framework I/O: <http://www.kegel.com/c10k.html>

In tal modo il server impiega la maggior parte del proprio tempo attendendo che i processi bloccati si completino, e questo porta ad una vistosa inefficienza nello sfruttamento delle risorse a disposizione.

Nell'esempio in figura di seguito<sup>68</sup>, notiamo la differenza tra una semplice applicazione PHP ed una sviluppata con Node:



Questo particolare approccio è fondamentale nel sistema che si sta realizzando per questo lavoro di tesi poiché le chiamate da diversi client avvengono in maniera asincrona e necessitano di una risposta in tempo reale; in altre parole non è ammissibile un'attesa del completamento di tutti i processi prima delle risposte dal server.

I moduli principali forniti dal framework sono illustrati di seguito. Saranno approfonditi solo quelli basilari che rientrano in qualsiasi applicazione web e che verranno utilizzati anche nel sistema.

- **globals:** il modulo globals non rappresenta una vera e propria libreria ma una serie di API incluse nel namespace globale dell'applicazione e quindi richiamabili direttamente. Tra le funzioni di questo pseudo-modulo, le principali sono:
  - require: che permette di includere moduli aggiuntivi;
  - setTimeout/setInterval: per lavorare con i timer.

Oltre alle funzioni sono presenti alcuni oggetti fondamentali come “console” che permette di accedere allo standard “input” e allo standard “error” del processo e alcune variabili valorizzate a run-time.

- **http:** Include la classe “bootstrap” per avviare un server web e una serie di oggetti che permettono di utilizzare il protocollo in maniera più semplice ad un livello di astrazione più alto.

---

<sup>68</sup> Figura estratta da: [http://www.mrwebmaster.it/javascript/introduzione-node-js\\_12163.html](http://www.mrwebmaster.it/javascript/introduzione-node-js_12163.html)

- **url:** Questo modulo permette di lavorare con gli URL, creandoli a partire da oggetti oppure trasformandoli in oggetti a partire da stringhe. Di fondamentale importanza è infatti il metodo “parse” che trasforma un URL in un oggetto.
- **path:** Il modulo path permette di lavorare con i percorsi reali della macchina sulla quale è avviata l'applicazione. Mette a disposizione metodi per navigare tra le cartelle e per comporre path complessi a partire da più stringhe.
- **FS – File System:** Questo modulo permette, insieme al precedente, di lavorare con il filesystem della macchina, eseguendo tutte le operazioni tipiche sui file e le cartelle.
- **util:** Include una serie di funzioni di utilità di gestione delle variabili e delle strutture dati.

Tutti i moduli necessari allo sviluppo del proprio sistema possono essere inclusi semplicemente grazie alla funzione globale “require”.

## 6.2. Persistenza: MongoDB - Una soluzione NoSQL

NoSQL è un movimento che promuove sistemi software caratterizzato dallo sviluppo di sistemi in cui la persistenza dei dati non utilizza il modello relazionale, di solito usato dai database tradizionali (RDBMS - Relational Data Base Management Systems). L'espressione “NoSQL” coniata nel 1998 in riferimento ad una base di dati relazionale open source che non usava un'interfaccia SQL si contrappone proprio a quest'ultimo, il più comune linguaggio di interrogazione dei dati nei database relazionali, preso a simbolo dell'intero paradigma relazionale.

Gli archivi NoSQL non richiedono uno schema fisso (si dicono schemaless), evitano spesso le operazioni di giunzione (join) e mirano alla scalabilità orizzontale.

All'opposto di quanto si potrebbe pensare, il termine NOSQL è acronimo di Not Only SQL, il che sta a significare che il movimento non può essere definito come contrario all'utilizzo di database relazionali.

Il nome era un tentativo per etichettare il crescente numero di data base non relazionali e distribuiti che spesso non forniscono le classiche caratteristiche ACID: atomicità, consistenza, isolamento, durabilità. Il motivo per il quale tali caratteristiche non venivano fornite è il cosiddetto Teorema CAP o teorema di Brewer<sup>69</sup> il quale afferma che è impossibile

<sup>69</sup> Riferimenti:

- Nancy Lynch and Seth Gilbert, “Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services”, ACM SIGACT News, Volume 33 Issue 2 (2002), pg. 51-59
- "Brewer's CAP Theorem", julianbrowne.com, Retrieved 02-Mar-2010
- "Brewers CAP theorem on distributed systems", royans.net
- Eric Brewer, "Towards Robust Distributed Systems"

per un sistema informatico distribuito fornire simultaneamente tutte e tre le seguenti garanzie:

- **Coerenza** (tutti i nodi vedano gli stessi dati nello stesso momento)
- **Disponibilità** (la garanzia che ogni richiesta riceva una risposta su ciò che è riuscito o fallito)
- **Tolleranza di partizione** (il sistema continua a funzionare nonostante arbitrarie perdite di messaggi)

Secondo il teorema, un sistema distribuito è in grado di soddisfare al massimo due di queste garanzie allo stesso tempo, ma non tutte e tre.

La base di dati scelta per la persistenza è quindi MongoDB per motivi di scalabilità e previsione dell'espansione del sistema a livello globale in più lingue, il che richiederebbe diversi server e database ampiamente distribuiti.

Le caratteristiche dei database NoSQL, rappresentano un vantaggio rispetto ai database SQL-like, tra questi, i principali sono identificabili in:

- **Leggerezza computazionale:** i database NoSQL non prevedono operazioni di aggregazione sui dati, in quanto tutte le informazioni sono già raccolte in un unico documento associato all'oggetto da trattare. Negli ambienti SQL la complessità di queste operazioni, e quindi il peso computazionale, cresce con la base di dati, del numero di tabelle e delle informazioni da trattare. Il NoSQL, invece, non ha limiti di dimensioni in questo senso. Così si ottengono migliori prestazioni e performance anche in ambienti di Big Data. Il prezzo da pagare a tutta questa flessibilità e alla proprietà di aggregazione dei database NoSQL è la duplicazione delle informazioni. In realtà, i costi sempre meno proibitivi dei sistemi di storage rendono questo svantaggio poco importante.
- **Assenza di schema:** i database NoSQL sono privi di schema in quanto il documento JSON contiene tutti i campi necessari, senza necessità di definizione. In questo modo, è possibile arricchire le applicazioni di nuovi dati e informazioni senza rischi per l'integrità dei dati. I database non relazionali, a differenza di quelli SQL, si rivelano quindi adatti a inglobare velocemente nuovi tipi di dati e a conservare dati semi-strutturati o non strutturati.
- **Scalabilità orizzontale:** l'aggregazione dei dati e l'assenza di uno schema definito a priori offre l'opportunità di scalare orizzontalmente i database NoSQL senza difficoltà e senza rischi operativi.

MongoDB<sup>70</sup> è definito come database NoSQL orientato al documento (document oriented o più brevemente doc-oriented), implementato in C++. Una Base di dati orientata al documento è un'ottima scelta per applicazioni orientate al documento.

Le basi di dati orientate al documento non memorizzano i dati in tabelle con campi uniformi per ogni record come nei database relazionali, ma ogni record è memorizzato come un documento che possiede determinate caratteristiche. Qualsiasi numero di campi con qualsiasi lunghezza può essere aggiunto al documento e ogni campo può anche contenere pezzi multipli di dati.

E' possibile implementare database doc-oriented realizzando uno strato software da porre su un database ad oggetti relazionale (ORDBMS) oppure partendo da un database orientato agli oggetti (OODBMS). Nel primo caso gli oggetti prenderanno posto direttamente nel database e saranno resi disponibili attraverso query in un apposito linguaggio. MongoDB segue il secondo approccio, quello in cui si utilizzano oggetti persistenti in un linguaggio di programmazione orientato agli oggetti ed un set di API per memorizzare e recuperare gli oggetti stessi.

MongoDB predilige l'incorporamento piuttosto che la normalizzazione. In un modello relazionale sinonimo di qualità di un database è il livello di normalizzazione adottato per le sue tabelle (salvo poi rinunciarvi a favore di maggiori performance). Proprietà di una certa rilevanza per l'applicativo vengono quindi solitamente promosse ad Entità separate, referenziate con l'Entità originaria, in modo da non perderne semantica e contenuto informativo. Nei doc-oriented si segue un indirizzo opposto, accorpando quanto più possibile gli oggetti, creando macro-entità dal massimo contenuto informativo, che risucchiano/incorporano tutte le informazioni di cui necessitano per una determinata semantica.

Come in tutti i DBMS, anche ai doc-oriented è richiesta la presenza di indici. In MongoDB:

- il campo `_id` è indicizzato automaticamente
- i campi sui quali è tipico eseguire ricerche o acceduti di frequente andrebbero indicizzati
- i campi su cui sono definiti ordinamenti generalmente andrebbero indicizzati

MongoDB fornisce strumenti in grado di suggerire su quali campi sia opportuno definire indici, dove non ve ne fossero.

Una delle più importanti caratteristiche di MongoDB è la sua capacità di supportare query dinamiche (ad hoc). I sistemi che supportano le query dinamiche non richiedono degli indici speciali per trovare i dati; gli utenti possono trovare i dati usando un qualsiasi criterio.

---

<sup>70</sup> Sito MongoDB: <https://www.mongodb.com/>

MongoDB supporta una vasta gamma di selettori per i documenti. Tra le possibili query effettuabili notiamo:

- **Selettore di campo:** è possibile specificare di mostrare nei risultati solo un sottoinsieme di campi. In maniera analoga a quanto avviene nella SELECT di SQL, l'unica differenza è che il campo id viene sempre restituito, anche quando non specificato.
- **Selezione:** è possibile indicare criteri attraverso i quali recuperare un sottoinsieme dei documenti in una collection. In maniera affine a quanto si fa in SQL con la WHERE. E' possibile ovviamente utilizzare per la selezione, come in SQL:
  - operatori condizionali (and, or, nor, in, nin, all, exists, mod, size, type, ne)
  - espressioni regolari
  - valori in array
  - valori negli oggetti embedded
  - meta operatori (not)
  - operatore di aggregazione (group)
- **Ordinamenti:** le query in MongoDB possono ritornare risultati ordinati in maniera crescente o decrescente, specificando opportunamente su quale campo eseguire l'ordinamento. In maniera analoga a quanto avviene con la clausola ORDER BY in SQL.
- **Skip & Limit:** sono le opzioni tramite le quali effettuare la paginazione dei risultati in maniera semplice.

### 5.3. Relatore e Partecipanti - Socket.IO<sup>71</sup> per il broadcasting

Nel contesto di lavoro in cui si colloca il sistema, è necessario che il relatore di un evento sia capace di inviare in maniera trasparente, rapida ed efficiente il testo trascritto dal parlato ai dispositivi dei partecipanti.

Socket.IO è una libreria JavaScript creata da Guillermo Rauch per applicazioni web real time che permette la comunicazione in tempo reale bidirezionale tra web client e server. La libreria è composta da due sottosezioni: la parte client che va in esecuzione sul browser e la parte server che gira proprio sulla macchina server con Node.js. Entrambe le componenti hanno API identiche e proprio come Node, funziona in maniera event-driven offrendo quindi migliori prestazioni.

---

<sup>71</sup> Socket.IO: <http://socket.io/>

Socket.IO mira a rendere possibile la realizzazione di applicazioni in tempo reale in tutti i browser e dispositivi mobili, senza fare confusione con le differenze tra i diversi meccanismi di trasporto.

Alcune delle caratteristiche principali della libreria sono:

- Supporto trasporti multipli (browser datati, browser per dispositivi mobile, ecc.)
- Socket multiple sotto la stessa connessione.
- Rilevamento disconnessione attraverso heartbeats (letteralmente “battiti cardiaci”).
- Supporto riconnessione con il buffering (ideale per i dispositivi mobili).
- Protocollo leggero che si trova sopra il protocollo http.

Al fine di fornire prestazioni degne di un sistema che lavora in tempo reale su tutti i browser, Socket.IO seleziona il metodo di trasporto migliore tra a run-time tra Websocket, Adobe Flash Socket, AJAX long polling, AJAX multipart streaming, Forever Iframe, JSONP Polling, senza impattare sull’uso delle API.

Una feature fondamentale e decisiva per la scelta di questa libreria, è stata la semplicità con cui è possibile inviare messaggi broadcast a tutti i client connessi da dispositivi e browser differenti alla socket server, in esecuzione sul server Node, in maniera rapida ed efficiente.

## 6.4. Applicazione Web - Cordova per lo sviluppo cross-platform

Allo stato attuale dell'avanzamento tecnologico in ambito mobile, si deve necessariamente considerare la necessità di sviluppare sistemi che siano eseguibili anche su tali dispositivi. Al momento chi vuole creare e pubblicare un'app ha di fronte almeno due possibilità: imparare a sviluppare su una specifica piattaforma e rilasciare la propria app soltanto per essa o imparare a lavorare su più piattaforme per garantirsi un pubblico più vasto. Considerata la rapida evoluzione dei sistemi che costringe ad una formazione permanente, imparare a sviluppare su più piattaforme è un'impresa piuttosto ardua. Di fronte a questo dilemma può essere utile valutare una terza possibilità che consente di avere una visione unificata delle diverse piattaforme, una strada che passa per le tecnologie Web.

Allo stato attuale possiamo classificare le applicazioni per il mondo mobile in base alle tecnologie utilizzate in tre categorie:

- **app native:** scritte e compilate per una specifica piattaforma utilizzando uno dei linguaggi di programmazione supportati dal particolare sistema operativo. Ovviamente offre una maggiore velocità di esecuzione e una maggiore integrazione con il look della piattaforma, ma non è direttamente portabile su altre piattaforme
- **web app:** fondamentalmente pagine Web ottimizzate per dispositivi mobili sfruttando tecnologie come HTML5, JavaScript e CSS. Una Web App risulta

indipendente dalla piattaforma ma richiede una connessione Internet costantemente attiva e non è in grado di accedere al filesystem o ad altre risorse hardware del dispositivo

- **app ibride:** cercano di sfruttare il meglio delle due categorie precedenti, scritte con tecnologie Web ma eseguite localmente all'interno di un'applicazione nativa. Le App ibride coniugano i vantaggi delle web app con quelle delle app native, consentendo di utilizzare le tecnologie Web per sviluppare l'applicazione senza necessità di una connessione Internet costante e avendo accesso alle risorse locali del dispositivo.

Anche in questo caso però ci sono degli svantaggi da considerare:

- una minore efficienza nel rendering grafico;
- la potenziale lentezza di esecuzione nell'accesso alle risorse locali;
- senza particolari accorgimenti l'aspetto dell'interfaccia grafica potrebbe non risultare abbastanza omogeneo con quello nativo della piattaforma.

Apache Cordova<sup>72</sup> nasce dal progetto PhoneGap, avviato da una azienda canadese, Nitobi Software, e venduto nel 2011 ad Adobe.

Si tratta di un software Open Source distribuito con licenza Apache 2.0. L'architettura di Apache Cordova si presenta come una sorta di contenitore di applicazione Web eseguita localmente. L'interfaccia grafica di un'applicazione Cordova è infatti costituita da una Web view che occupa l'intero schermo del dispositivo, all'interno della quale viene visualizzato l'HTML ed il CSS ed eseguito il codice JavaScript. Tramite JavaScript è possibile accedere ad un ricco insieme di API che interfacciano l'applicazione Web con le funzionalità della piattaforma ospite. Web view e API sono le componenti dell'applicazione che dipendono dalla specifica piattaforma mobile e sono le componenti che il framework mette a disposizione dello sviluppatore, consentendo di concentrarsi su del codice indipendente dalla piattaforma. Cordova implementa infatti lo stesso insieme di API sulle diverse piattaforme mobile supportate creando un livello software standard a cui si possono interfacciare le diverse applicazioni sviluppate. Attualmente Apache Cordova supporta le seguenti piattaforme mobile: Android, iOS, Blackberry, Bada, Tizen e Windows Phone.

PhoneGap, così come Ionic (che sarà approfondito successivamente), Monaca, TACO, Intel XDK e Telerik, lavorano attualmente sulla base di Cordova e da questa dipende il loro funzionamento.

Apache Cordova può essere esteso con diversi plug-in permettendo agli sviluppatori di aggiungere le più disparate funzionalità. Adobe Systems avvisa gli sviluppatori che le applicazioni ibride costruite con Cordova potrebbero essere respinte da Apple perché considerate troppo lente o non appropriate per l'interfaccia. Questo può essere un problema

---

<sup>72</sup> Apache Cordova: <https://cordova.apache.org/>

molto importante che potrebbe essere aggirato grazie ad una buona progettazione e al supporto di tecnologie che saranno illustrate successivamente.

## 5.5. Web Client - Angular 2<sup>73</sup>: Un framework Mobile e Desktop

Per avere un sistema scalabile dalle buone performance, è necessario adottare un framework client side che possa offrire alte prestazioni. Tra i numerosi framework esistenti, uno tra tutti si è distinto per le qualità che mostra.

AngularJS è un framework JavaScript per lo sviluppo di applicazioni Web client side nato da Misko Hevery e Adam Abrons. La versione 1.0 è stata rilasciata nel 2012, il progetto ha riscosso un notevole successo dovuto all'approccio di sviluppo proposto e all'infrastruttura fornita che incoraggia l'organizzazione del codice e la separazione dei compiti nei vari componenti. Angular inoltre è un progetto supportato da Google, il che ha dato una forte spinta al suo avanzamento e al suo successo.

AngularJS è un framework, ovvero una infrastruttura per la creazione di applicazioni composta da un insieme di funzionalità. Citando la documentazione ufficiale: Angular è quello che HTML avrebbe dovuto essere se fosse stato progettato per sviluppare applicazioni. Per raggiungere questo obiettivo, AngularJS da un lato esalta e potenzia l'approccio dichiarativo dell'HTML nella definizione dell'interfaccia grafica, dall'altro fornisce strumenti per la costruzione di un'architettura modulare e testabile della logica business di un'applicazione.

Angular fornisce tutto ciò che occorre per creare applicazioni moderne che sfruttano le più recenti tecnologie, come ad esempio le Single Page Application, cioè applicazioni le cui risorse vengono caricate dinamicamente su richiesta, senza necessità di ricaricare l'intera pagina. Tra le principali funzionalità a supporto dello sviluppo di tali applicazioni, è possibile individuare:

- Binding bidirezionale (two-way binding)
- Dependency injection
- Supporto al pattern MVC
- Supporto ai moduli
- Separazione delle competenze
- Testabilità del codice
- Riusabilità dei componenti

Nell'ottobre 2014, è giunto l'annuncio dell'inizio dei lavori per Angular 2. Contrariamente a quanto ci si possa aspettare, tale annuncio ha scatenato non poche polemiche non solo da

---

<sup>73</sup> AngularJS: <https://angularjs.org/>

parte di chi guardava il framework con una certa diffidenza, ma anche da parte di chi lo aveva apprezzato e utilizzato. Oggetto delle critiche era il fatto che Angular 2 sarebbe stata una completa riscrittura del framework senza alcuna compatibilità con la versione precedente:

*“Il nostro obiettivo con Angular 2 è di costruire il miglior insieme di strumenti per creare applicazioni Web senza essere vincolati a mantenere una retrocompatibilità con le API esistenti.”*

*(ng-europe, Angular 1.3, and beyond)*

Ad oggi però, è possibile apprezzare appieno i vantaggi che ha portato la riscrittura del framework in termini di prestazioni e facilità di utilizzo.

La riscrittura di Angular è stata guidata da alcuni principi fondamentali che il team di sviluppo si è sforzato di seguire:

- **Proiezione nel futuro:** Angular 2 è pensato per supportare tutte le novità di ECMAScript 2015 ed è esso stesso scritto seguendo le ultime specifiche del linguaggio. Tuttavia sono supportate anche le specifiche ES5 di JavaScript in modo da rendere possibile un avvicinamento graduale alla nuova sintassi. Il framework supporta inoltre anche altri linguaggi come TypeScript e Dart.
- **Mobile first:** uno degli obiettivi del nuovo Angular è di proporsi per lo sviluppo di applicazioni Web sia per l'ambiente desktop che per il mobile. Angular 2, infatti, supporta di default eventi touch e gesture e promette elevate prestazioni per assicurare una interazione fluida sui dispositivi mobile.
- **Riduzione della curva di apprendimento:** una delle principali critiche mosse alla precedente versione di Angular era la complessità del framework e la notevole mole di concetti che lo sviluppatore doveva comprendere. Era inoltre possibile fare la stessa cosa in modi diversi e non sempre era semplice comprendere quale fosse l'approccio migliore. Angular 2 ha provato a semplificare le cose e sono stati eliminati concetti come i controller e gli scope, riducendo il numero di direttive e focalizzando l'attenzione verso l'elemento chiave di dell'interfaccia Web di un'applicazione: il componente.
- **Modularità:** l'architettura di Angular 2 è altamente modulare e favorisce la scrittura di applicazioni a loro volta modulari. La maggior parte dei componenti del framework è opzionale ed è possibile sostituirli con altri di terze parti. Questo aspetto mitiga una delle principali limitazioni di Angular 1.x, ovvero l'integrazione con librerie esterne o con altri framework.

- **Prestazioni:** un'attenzione particolare è stata posta sulle prestazioni del framework, dalla riduzione dei tempi di caricamento e bootstrapping dell'applicazione al rilevamento efficiente delle modifiche nel modello dei dati ed alla velocità dei tempi di rendering. Il risultato di questa attenzione è rilevabile da alcuni benchmark<sup>74</sup> effettuati da terze parti.

## 6.6. App ibride e Performance - Ionic 2: Un framework omnicomprensivo

Ionic<sup>75</sup> è definito come un SDK (Software Development Kit) open source (sotto licenza MIT) per lo sviluppo di app mobile ibride. Ionic è costruito sulla base di AngularJS e Apache Cordova e mette a disposizione strumenti e servizi per lo sviluppo rapido di app mobile ibride usando HTML5, CSS e Sass (che sarà brevemente descritto in seguito). Ionic è stato creato nel 2013 da Max Lynch, Ben Sperry e Adam Bradley della Drifty Co. I prodotti principali dell'azienda appena citata sono Codiqa and Jetstrap, cioè strumenti drag-and-drop-based per la creazione di interfacce usando jQuery Mobile e Bootstrap. Grazie al feedback positivo dei clienti che hanno utilizzato tali strumenti, il team ha deciso di sviluppare il proprio framework che sarebbe stato finalizzato al miglioramento delle performance utilizzando i più moderni standard Web. Gli autori hanno raccolto le migliori best practice per lo sviluppo di interfacce mobile con tecnologie Web e le hanno codificate in questo framework, evitando quindi che ogni sviluppatore dovesse ripartire da zero nello sviluppo dell'interfaccia di una nuova applicazione.

Nel 2015, alla data di uscita della versione stabile, dopo diversi rilasci Alpha e Beta, i diversi sviluppatori che hanno utilizzato Ionic sono riusciti a creare più di 1.3 milioni di app con il nuovo SDK.

Il framework Ionic fornisce tutte le funzionalità classiche che possono essere trovate in un SDK per lo sviluppo mobile oltre a componenti predefiniti per le diverse piattaforme mobile, paradigmi interattivi e temi di base estensibili dal programmatore. Essendo costruito su Angular, Ionic fornisce inoltre componenti e metodi per interagire con esso. Oltre a tutto questo, Ionic permette di utilizzare funzioni come le notifiche push, A/B testing, deploy del codice e build automatiche per ciascuna piattaforma grazie al supporto di Cordova e la CLI (command-line interface) fornita.

<sup>74</sup> Benchmark Angular 2:

- <http://www.stefankrause.net/wp/?p=191>
- <https://autho.com/blog/more-benchmarks-virtual-dom-vs-angular-12-vs-mithril-js-vs-the-rest/>

<sup>75</sup> Ionic framework: <http://ionicframework.com/>

Rispetto alle applicazioni ibride più classiche, mescolare codice Ionic con codice di app mobile native, porta ad un prodotto con prestazioni più elevate del normale. Utilizzando AngularJS (invece di jQuery) Ionic fa affidamento sulla accelerazione hardware nativo (piuttosto che la manipolazione del DOM). Ionic utilizza le transizioni CSS e trasformazioni per l'animazione sfruttando la GPU e massimizzare il tempo del processore disponibile.

Per la gestione del look'n'feel, Ionic utilizza SASS. L'aspetto grafico predefinito è ispirato ad iOS7, ma è possibile effettuare delle personalizzazioni effettuando l'override del CSS e/o impostando variabili in SASS (Syntactically Awesome Stylesheets) ovvero un precompilatore che permette di incrementare le funzionalità garantite dai CSS.

Lo sviluppo di applicazioni ibride con Ionic è semplificato dalla disponibilità di un vero e proprio ambiente di sviluppo che consente di creare lo scheletro iniziale di un'applicazione, di compilarla per una piattaforma mobile e di effettuare il debug all'interno di un browser.

Oggi Ionic è il framework per costruire applicazioni mobile cross-platform più popolare al mondo e diverse aziende in crescita e startup lo utilizzano in maniera efficace.

Il team di Ionic tuttavia annuncia:

*"But we aren't satisfied with Ionic yet. We think it can be so much more for so many more people. We want to enable not just the current generation of mobile developers, but the millions more that are just realizing they can build for these amazing devices, too.*

*With Ionic 2, we've gone back to square one and completely rethought how a mobile toolkit should work. We've dramatically improved performance, reduced the complexity of the code you need to write, and made it possible to build completely bespoke apps that compete with the best apps in the world."*

Inutile aggiungere altro, il team di Ionic promette che il nuovo framework in via di rapido sviluppo, sarà in grado di fornire performance notevolmente migliori riducendo la complessità del codice richiesto. Al momento Ionic 2 è in versione Beta ed è stato deciso di adottarlo in questo progetto sia per sfruttare al massimo le prestazioni che è in grado di raggiungere, sia per supportarne e incoraggiarne lo sviluppo.

## 7. Sprint 2 – Primo spike di ASR e Broadcast

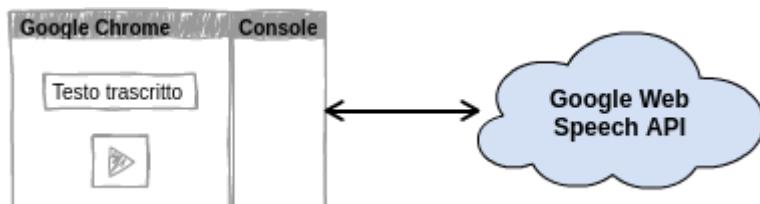
Al termine dello sprint precedente, dopo l'accettazione da parte del team delle scelte effettuate, sono stati programmati i primi task di "spike" delle tecnologie. In primo luogo sarebbe dovuto essere testato il sistema di riconoscimento del parlato (ASR System) individuato, ossia Google Web Speech API; in secondo luogo si sarebbe dovuto testare il sistema di produzione e broadcast di messaggi dal sistema che serve il relatore ai sistemi che servono i partecipanti (con SocketIO).

In questo capitolo sarà mostrata inoltre l'architettura di base che compone il sistema che si sta realizzando, man mano che si procederà con gli sprint e di conseguenza con lo sviluppo, saranno spiegati meglio i dettagli implementativi.

### 7.1. Primo test Google Web Speech API

Per verificare grossolanamente l'accettabilità del servizio e le sue prestazioni, è stato predisposto un semplice test grezzo: una pagina web in HTML che visualizza i risultati parziali e definitivi della trascrizione dopo la pressione di un bottone per l'avvio della trascrizione stessa.

Per verificare il log, ci si è serviti della console di Chrome attivabile tramite la pressione del tasto F12.



Il codice per utilizzare il servizio è stato recuperato dal codice di esempio fornito da Google e arricchito di messaggi di Log per verificarne il comportamento.

#### Transcriptor.js

```
var final_transcript = '';
var recognizing = false;

if ('webkitSpeechRecognition' in window) {
    var recognition = new webkitSpeechRecognition();
    recognition.continuous = true;
    recognition.interimResults = true;

    recognition.onstart = function () {
        recognizing = true;
        console.log("RECOGNITION STARTED");
    }

    recognition.onerror = function (error) {
        console.log("RECOGNITION ERROR: " + error.message);
    }

    recognition.onresult = function (event) {
        if (event.results.length > 0) {
            var transcript = event.results[0][0].transcript;
            if (transcript !== final_transcript) {
                final_transcript = transcript;
                console.log("RECOGNITION RESULTS: " + transcript);
            }
        }
    }
}
```

```

};

recognition.onerror = function (event) {
    console.log("RECOGNITION ERROR: " + event.error);
    recognition.start();
};

recognition.onend = function () {
    console.log("RECOGNITION STOPPED");
    if(recognizing){
        recognition.start();
        console.log("RECOGNITION RESTARTED");
    }
};

recognition.onresult = function (event) {
    var interim_transcript = '';
    for (var i = event.resultIndex; i < event.results.length; ++i) {
        if (event.results[i].isFinal) {
            final_transcript += event.results[i][0].transcript;
            console.log("CONFIDENCE (" +
event.results[i][0].transcript +
") = " +
event.results[i][0].confidence);
        } else {
            interim_transcript += event.results[i][0].transcript;
        }
    }
    final_transcript = capitalize(final_transcript);
    final_span.innerHTML = linebreak(final_transcript);
    interim_span.innerHTML = linebreak(interim_transcript);
};

recognition.onaudiostart= function (event) {
    console.log("AUDIO START");
};

recognition.onsoundstart= function (event) {
console.log("SOUND START");
};

recognition.onspeechstart= function (event) {
console.log("SPEECH START");
};

recognition.onspeechend= function (event) {
    console.log("SPEECH END");
};

recognition.onsoundend= function (event) {
    console.log("SOUND END");
};

recognition.onnomatch= function (event) {
    console.log("NO MATCH");
};

};

var two_line = /\n\n/g;
var one_line = /\n/g;
function linebreak(s) {

```

```

        return s.replace(two_line, '<p></p>').replace(one_line, '<br>');
    }

function capitalize(s) {
    return s.replace(s.substr(0, 1), function (m) {
        return m.toUpperCase();
    });
}

function startDictation(event) {
    if (recognizing) {
        recognition.stop();
        recognizing=false;
        start_button.innerHTML = "START"
        return;
    }
    final_transcript = '';
    recognition.lang = 'it-IT';
    recognition.start();
    start_button.innerHTML = "STOP"
    final_span.innerHTML = '';
    interim_span.innerHTML = '';
}

```

Il codice riportato costituisce il servizio di trascrizione del parlato. Tutto è basato sull'oggetto “recognition” di tipo “webkitSpeechRecognition” disponibile in Google Chrome. I metodi riportati, per la maggior parte costituiscono listener di eventi:

- **onstart**: ascolta l'evento di inizio del riconoscimento e imposta i flag che identificano che il riconoscimento è in corso.
- **onerror**: ascolta il lancio di un errore da parte del servizio di riconoscimento e lo stampa in console.
- **onend**: ascolta l'arresto del riconoscimento e imposta i flag che identificano che il riconoscimento non è attivo.
- **onresult**: ascolta la restituzione di risultati di tipo parziale o finale da parte del servizio di riconoscimento. Imposta le variabili apposite il cui contenuto verrà stampato sulla pagina HTML.
- **onaudiostart**: ascolta l'evento di ricezione di audio dal dispositivo di input predefinito e stampa in console che l'evento si è verificato.
- **onsoundstart**: ascolta l'evento di ricezione di suono dal dispositivo di input predefinito e stampa in console che l'evento si è verificato.
- **onspeechstart**: ascolta l'evento di ricezione di parlato dal dispositivo di input predefinito e stampa in console che l'evento si è verificato.

- **onspeechend**: ascolta l'evento di termine del parlato dal dispositivo di input predefinito e stampa in console che l'evento si è verificato.
- **onsoundend**: ascolta l'evento di termine di ricezione di suono dal dispositivo di input predefinito e stampa in console che l'evento si è verificato.
- **onnomatch**: ascolta l'evento lanciato dal servizio di riconoscimento che indica che non è stato trovato una corrispondenza testuale per il parlato ricevuto. Stampa in console che l'evento si è verificato.

Non tutti gli eventi saranno ascoltati nella versione definitiva, in questo caso sono stati inseriti per verificare il comportamento del sistema in diverse condizioni.

Le funzioni di supporto seguenti servono per formattare il testo che verrà stampato sulla pagina HTML:

- **linebreak**: manda a capo il testo.
- **capitalize**: rende maiuscola la prima lettera del testo.

La funzione “**startDictation**” infine, sarà chiamata dal bottone per avviare o fermare il riconoscimento vocale.

Lo spike è stato effettuato scegliendo innanzitutto del testo che un utente umano avrebbe dovuto leggere dopo aver attivato il riconoscimento. Il servizio di trascrizione avrebbe stampato il testo sulla pagina HTML oltre a tutti gli eventi verificatisi in console. La verifica della buona riuscita dello spike è stata effettuata confrontando parola per parola il testo letto con quello trascritto.

Il test è stato condotto in diverse condizioni: in ambiente rumoroso e in ambiente silenzioso per verificare le differenze di prestazioni del servizio dovute al contesto in cui esso viene utilizzato.

Dopo aver effettuato il test, è stato riscontrato che la percentuale di testo riconosciuto correttamente dal servizio si aggira intorno all’84% in ambiente completamente silenzioso e intorno al 76% in ambiente rumoroso (in strada poco trafficata). In condizioni di silenzio, è stato rilevato che il restante 16% era costituito da parole rilevate con genere invertito (maschile/femminile); connettivi saltati (e, il, con, da...) e poche parole completamente errate nella semantica ma che nel contesto erano facilmente individuabili; o nomi propri di persona errati attribuibile al vocabolario disponibile. Durante le pause nel parlato, spesso succedeva che l’assenza di audio comportava uno stop automatico del servizio; il problema è stato poi risolto inserendo un comando di riavvio automatico del servizio quando questo non fosse stato arrestato esplicitamente dall’utente.

In condizioni rumorose le percentuali di parole errate nel genere aumenta leggermente così come quella di parole errate nella semantica. Questo problema potrebbe essere momentaneamente ignorato se si considera che il contesto in cui il sistema dovrebbe

funzionare è caratterizzato da silenzio di fondo o leggero brusio che non dovrebbe creare rumore se si utilizza un microfono direzionale come dispositivo di input. La problematica piuttosto rilevante è invece quella che diversi sviluppatori che utilizzano lo stesso servizio hanno riscontrato senza trovarne una soluzione valida, ovvero il livelock del riconoscimento durante l'esecuzione: il servizio risulta attivo e nessun errore viene lanciato, tuttavia non viene ricevuto nessun risultato parziale né definitivo. Dopo diverse prove, è stato rilevato che probabilmente il problema è attribuibile alla sovrapposizione di voci differenti durante il riconoscimento. Probabilmente il servizio cerca di riconoscere le voci senza successo. Il problema potrebbe essere risolto impostando un time-out entro il quale, se non si ricevono risultati, il servizio viene riavviato. Questo comporterebbe una perdita della trascrizione in quel lasso di tempo che però potrebbe essere accettabile considerato che i destinatari della trascrizione sono utenti umani che sarebbero quindi in grado di interpolare le parole mancanti intuendo la semantica corretta. Un'altra soluzione potrebbe essere l'interpolazione automatica delle parole mancanti ricorrendo ad un sistema di Natural Language Processing (NLP). Si è deciso di rimandare la risoluzione di questo problema ad una fase successiva dello sviluppo considerato che non ci sono alternative valide che potrebbero rimpiazzarlo. La considerazione più importante e da tener presente quando il sistema sarà utilizzato in contesti reali è che le parole dovrebbero essere ben scandite in modo da ridurre sensibilmente le possibilità di errore di riconoscimento.

### Testo Letto:

C'erano una volta tre pesci che vivevano in uno stagno: uno era intelligente, un altro lo era a metà e il terzo era stupido. La loro vita era quella di tutti i pesci di questo mondo, finché un giorno arrivò un uomo.  
L'uomo portava una rete e il pesce intelligente lo vide attraverso l'acqua. Facendo appello all'esperienza, alle storie che aveva sentito e alla propria intelligenza, il pesce decise di passare all'azione.  
“Dato che ci sono pochi posti dove nascondersi in questo stagno, farò finta di essere morto”, pensò. Raccolte tutte le sue forze, balzò fuori dall'acqua e atterrò ai piedi del pescatore, che si mostrò piuttosto sorpreso. Tuttavia, visto che il pesce tratteneva il respiro, l'uomo lo credette morto e lo ributtò nello stagno. Allora il nostro pesce si lasciò scivolare in una piccola cavità sotto la riva.  
Il secondo pesce, quello semi-intelligente, non aveva capito bene quanto era accaduto. Raggiunse quindi il pesce intelligente per chiedergli spiegazioni. “Semplice”, disse il pesce intelligente, “ho fatto finta di essere morto e così mi ha ributtato in acqua”. Immediatamente, il pesce semi-intelligente balzò fuori dall'acqua e cadde ai piedi del pescatore.  
“Strano”, pensò il pescatore, “tutti questi pesci che saltano fuori dappertutto!”. Ma il pesce semi-intelligente si era dimenticato di trattenere il respiro, così il pescatore si accorse che era vivo e lo mise nel suo secchio. Riprese quindi a scrutare la superficie dell'acqua, ma lo spettacolo di quei pesci che atterravano sulla riva, ai suoi piedi, lo aveva in qualche modo turbato, sicché si dimenticò di chiudere il secchio. Quando il pesce se ne accorse, riuscì faticosamente a scivolare fuori e a riguadagnare lo stagno a

piccoli salti. Andò a raggiungere il primo pesce e, ansimando, si nascose accanto a lui. Ora, il terzo pesce, quello Stupido, non era naturalmente in grado di trarre vantaggio dagli eventi, neanche dopo aver ascoltato il racconto del primo e del secondo pesce. Allora riesaminarono ogni dettaglio con lui, sottolineando l'importanza di non respirare quando si finge di essere morti.

"Molte grazie, adesso ho capito!"; disse il pesce stupido, e con quelle parole si lanciò fuori dall'acqua e andò ad atterrare proprio accanto al pescatore. Ora, il pescatore, che aveva già perso due pesci, lo mise subito nel secchio senza preoccuparsi di verificare se respirava o no. Poi lanciò ancora ripetutamente la sua rete nello stagno, ma i primi due pesci erano ormai al sicuro nella cavità sotto la riva. E questa volta il suo secchio era ben chiuso.

Il pescatore finì per rinunciare. Aprì il secchio, si accorse che il pesce stupido non respirava, lo portò a casa e lo diede da mangiare al gatto.

## Trascrizione e Log:

```

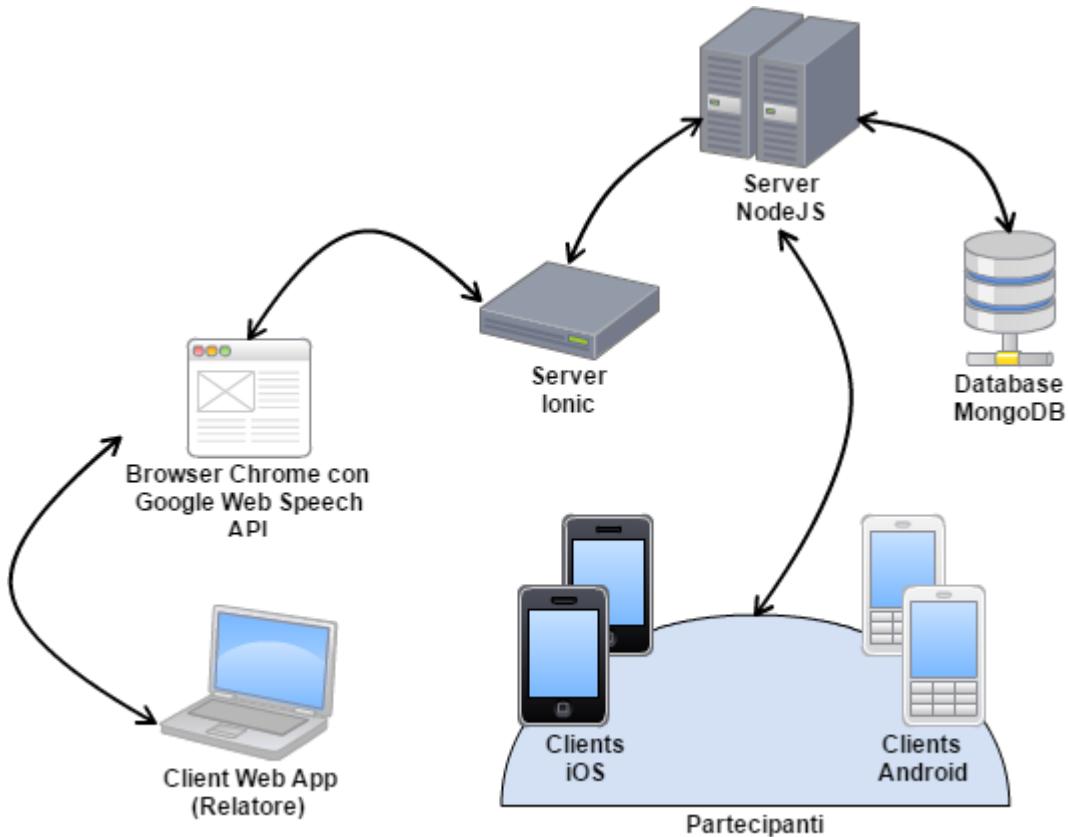
START
C'erano una volta tre pesci che vivevano nello stagno uno era intelligente un altro lo era a metà e il terzo era stupido la loro vita era quella di tutti i pesci che avevano finché un giorno arrivarono uno uomo e una donna una vera e propria intelligente li vide attraverso l'acqua facendo appello all'esperienza alle storie che aveva sentito e alle proprie intelligenza il pesce decise di passare all'azione dato che ci sono pochi posti dove nascondersi in questo stagno farà finita di essere vivo e potranno fuggire tutte le sue forze balzo fuori dall'acqua e torna ai piedi del pescatore che è molto più piuttosto sorpreso tuttavia visto che il pesce tranneva il respiro l'uomo lo credeva morto e lo ributto nello stagno allora il nostro pesce si lascia scivolare in una piccola cavità sottomarina il secondo pesce quello semintelligente non aveva capito bene quanto di creduto raggiunse quel pesce e gli parlò per chiedergli spiegazioni semplici di pesce intelligente ho fatto finta di essere morto e così mi hai buttato in acqua immediatamente il pesce semintelligente mangiò fuori dall'acqua e cade ai piedi del pescatore strano pensò il pescatore tutti questi pesci che saltano fuori dappertutto il pesce se mi intelligente si era dimenato di tutto per fuggire nello stagno e non era vero che il pesce era vivo e lo mise nel suo secchio ripreso e quindi ascoltava la superficie dell'acqua ma lo spettacolo di quei pesci che attiravano sulla riva ai suoi piedi lo aveva in qualche modo turbato si che si dimenchi se dimenico di chiudere il secchio quando il pesce se ne accorse riesci faticosamente a scivolare fuori e a saltare in acqua e torna ai suoi piedi saltando a raggiungere il primo pesce e ansimando si nascose accanto a lui ora il terzo pesce quello stupido non era naturalmente in grado di trarre vantaggio dagli eventi neanche dopo aver ascoltato il racconto del primo e del secondo pesce allora di esaminare ogni dettaglio con lui e sottolineando l'importanza di non respirare quando si finge di essere morti grazie adesso ho capito disse il pesce stupido e con quelle parole silenzio fuori dall'acqua e andò da Ferrara e propria carta pescatore ora il pescatore che aveva capito che aveva già preso due pesci lo vinse subito nel secchio senza preoccuparsi di verificare se respira va o no poi non so ancora ripetutamente la sua rete nello stagno e prima due pezzi erano ormai al sicuro nella cavità sotto la riva questa volta il suo secchio era ben chiuso il pescatore finì per rinunciare apri il secchio si accorse che il pesce stupido non respirava lo portò a casa e lo diede da mangiare al gatto

```

In conclusione, è possibile accettare appieno i piccoli difetti del servizio di trascrizione nella speranza di riuscire a risolvere quelli più grossi in maniera "elegante". Per un sistema di riconoscimento completamente automatico, infatti, le prestazioni sono ottime e le prospettive per il proseguo dello sviluppo, fermo restando che il servizio potrà essere sostituito successivamente, piuttosto buone.

## 7.2. Architettura del sistema

Stabilita la fattibilità del sistema visto che i risultati ottenuti dal primo spike di riconoscimento, ha seguito una progettazione grossolana dell'architettura del sistema (come in figura) che sarà descritta in questo paragrafo.



- **Il Database MongoDB:** come preaccennato, si tratta di una base di dati document-oriented NoSQL. Questo inizialmente per poter trarre vantaggio dalla scalabilità a livello di documento, in particolare sarà possibile aggiungere campi successivamente senza compromettere l'integrità. Il database conterrà tutte le informazioni relative agli utenti registrati, agli eventi con relative sessioni con relativi interventi.
- **Il Server NodeJS:** scritto in JavaScript contiene i business object di cui si avrà bisogno: utente, evento, sessione, intervento; si occuperà inoltre di ricevere tutte le richieste che perverranno dai client (relatori, organizzatori e partecipanti) per elaborare una risposta interfacciandosi con il database (unico componente in grado di farlo).
- **Il Server Ionic:** si tratta del web server che ospiterà l'elaborazione delle richieste da parte di un qualsiasi utente connesso alla Web App. Da solo contiene solo le informazioni di base per costruire l'interfaccia del sistema, invierà le richieste dei contenuti al server NodeJS e li elaborerà per mostrarli correttamente nell'interfaccia.
- **Il Browser:** il browser Google Chrome è di fondamentale importanza per poter utilizzare il servizio ASR, altrimenti inutilizzabile su altri browser. Tuttavia, le altre funzionalità come la gestione del profilo e degli eventi potrà essere tranquillamente effettuata su tutti gli altri browser.

- **Il Client principale (Web App):** Un qualsiasi dispositivo Desktop o Notebook dotato di connessione a internet, potrà collegarsi all'indirizzo su cui è ospitato il server Ionic per utilizzare il sistema (con Google Chrome in tutte le sue funzionalità). Importante sottolineare che anche i dispositivi mobili come tablet e smartphone saranno in grado di accedere al sistema via Web App, non saranno però in grado di supportare il riconoscimento del parlato in maniera corretta anche se si utilizza un browser Chrome. Su mobile, infatti, il servizio ASR si comporta in maniera insolita provocando un fastidioso glitch che trascrive più di una volta la stessa frase riconosciuta (il risultato della trascrizione dell'enunciato “prova” risulterà qualcosa di simile a “provaprovaaprova”).
- **I Client Mobile (Partecipanti):** i dispositivi smartphone ospiteranno l'applicazione mobile e tutta la logica di controllo come se integrasse il server Ionic, grazie alle potenzialità fornite da Ionic su Cordova. In particolare, l'app comunicherà direttamente con il server NodeJS per ottenere le informazioni necessarie a completare l'interfaccia. Il client Mobile (iOS o Android) sarà in grado di gestire eventi e profilo utente e di seguire un intervento in corso (visualizzare la trascrizione).

### 7.3. Primo Broadcast di messaggi

Una volta trascritti grazie alle Google Web Speech API, i messaggi devono essere inviati in tempo reale a tutti i client connessi. Per far ciò, ci si serve della libreria “socket.IO” descritta precedentemente.

Per questo primo spike, l'obiettivo è quello di riuscire a far comunicare il relatore con i client connessi. Allo scopo è stato predisposto un primo dispatcher scritto in JavaScript su NodeJS in grado di ricevere messaggi da un client identificato come “speaker” e trasmetterlo a tutti i client connessi in broadcast. Sono stati create quindi due pagine web HTML grezze: una associata al relatore che importa lo script di ASR (descritto precedentemente) e una associata al partecipante che importa soltanto la libreria socket.io client.

dispatcher.js

```
var http = require('http');

//Importa la libreria e inizializza la connessione sulla porta definita
var io = require('socket.io').listen(80);

//attiva il listener delle connessioni
io.sockets.on('connection', function (socket) {
```

```
//attiva il listener per un tipo di messaggio da parte di un client connesso
socket.on('client_message', function (data) {
    //emette il messaggio destinato a tutti i client connessi
    io.sockets.emit('server_message', { text: data.text });
});
});
```

Il modulo riportato lavora sul server NodeJS e si occupa di ricevere messaggi dal relatore e reinviarli a tutti i partecipanti.

### clientSide.html

```
<html>
  <head>
    <title>Partecipante</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <textarea id="msgArea"></textarea>
    <script type="text/javascript" src="http://localhost:80/socket.io/socket.io.js">
    </script>
    <script type="text/javascript">
        //Il client si conterà all'indirizzo
        var socket = io.connect('http://localhost:80');
        /*socket.on consente di ricevere I messaggi dal server */
        socket.on('server_message', function (data) {
            /* istruzione che consente di visualizzare sulla pagina html,
               nell'area denominata "msgArea", il messaggio del server*/
            document.getElementById("msgArea").innerHTML += (data.text);
        });
    </script>
  </body>
</html>
```

Questa pagina html incorpora la libreria (script) socket.io e riceverà messaggi dal modulo “dispatcher.js” iniettandoli nell’html.

### speakerSide.html

```
<html>
  <head>
    <title>Relatore</title>
    <script type="text/javascript" charset="utf-8" src=".transcriotor.js"></script>
  </head>
  <body>
    <div>
      <a href="#" id="start_button" onclick="startDictation(event)">Dictate</a>
    </div>
  </body>
</html>
```

```
<div id="results">
    <span id="final_span" class="final"></span>
    <span id="interim_span" class="interim"></span>
</div>
</body>
</html>
```

Questa pagina incorpora la libreria del servizio ASR (“transcriptor.js”) e contiene un bottone per avviare e fermare il servizio di riconoscimento del parlato. Inoltre contiene due span dedicati alla trascrizione parziale e definitiva in arrivo come risposta dal servizio di casa Google.

Il modulo “transcriptor.js” riportato sopra è stato semplicemente arricchito con le seguenti righe di codice necessarie rispettivamente per la connessione alla socket server e all’invio di messaggi ogni volta che si riceve un risultato definitivo dal servizio di riconoscimento.

```
var socket = io.connect('http://localhost:80');
socket.emit('client_type', {text: "Speaker"});
```

```
final_transcript += event.results[i][0].transcript;
socket.emit('client_message', {text: final_transcript});
```

Lo spike effettuato è stato organizzato in modo da verificare in quanto tempo i messaggi giungevano ai client e si è ottenuto un risultato molto confortante che vede giungere i risultati a destinazione mediamente in 2 secondi usando il server presente in laboratorio (COLLAB). Con rete 4G o Wi-Fi la differenza in termini di tempo è impercettibile, con connessione 3G invece, si può percepire un ritardo di circa mezzo secondo. Sia la connessione 4G, sia la 3G sono state testate utilizzando uno smartphone come hotspot.

## 8. Sprint 3 - Prima interfaccia Web e Mobile

Dopo l'accettazione dei task completati nello sprint precedente e la decisione comune del team che lo sviluppo del sistema poteva tranquillamente procedere, durante il meeting si è discusso delle successive azioni da intraprendere per continuare lo sviluppo.

Si è deciso di iniziare a sviluppare user story non più con tecnologie provvisorie (spike like) ma con le tecnologie scelte come definitive. In particolare sono stati programmati per lo sprint descritto in questo capitolo la user story “Login Utente” e un ulteriore spike di comunicazione tra il server Node e uno smartphone Android.

### 8.1. Comunicazione Server - Android

Lo spike di interazione tra server Node e Android, è stato necessario per capire se le prestazioni ottenute su Desktop, sarebbero state mantenute nella ricezione di messaggi da parte di uno smartphone. In questa fase non è stato ancora inizializzato il progetto Ionic ma è stato creato un nuovo progetto con Cordova utilizzando l'SDK Intel XDK. In altre parole, al nuovo progetto creato grazie alla procedura guidata del tool, è stata iniettata la libreria socket.io perché lo smartphone potesse ricevere i messaggi dal server. Il deploy è stato effettuato in maniera piuttosto semplice grazie al tool, è bastato infatti creare un certificato dummy per poter effettuare il deploy dell'app ed eseguire i due comandi seguenti dall'interfaccia di XDK.

```
cordova platforms add android  
cordova build android
```

Il risultato ottenuto è stato il file APK che è bastato trasferire e installare sul dispositivo mobile Android.

Il test è stato eseguito come in precedenza e i risultati ottenuti sono stati i medesimi.

#### 8.1.1. Intel XDK<sup>76</sup>

Un ambiente di sviluppo dedicato alle applicazioni ibride, che ultimamente è stato oggetto di grandi potenziamenti ed estensioni è Intel XDK.

Intel XDK è un ambiente di sviluppo gratuito e multi-piattaforma che permette a chiunque possieda competenze in HTML5, JavaScript e CSS di impiegarle velocemente nella

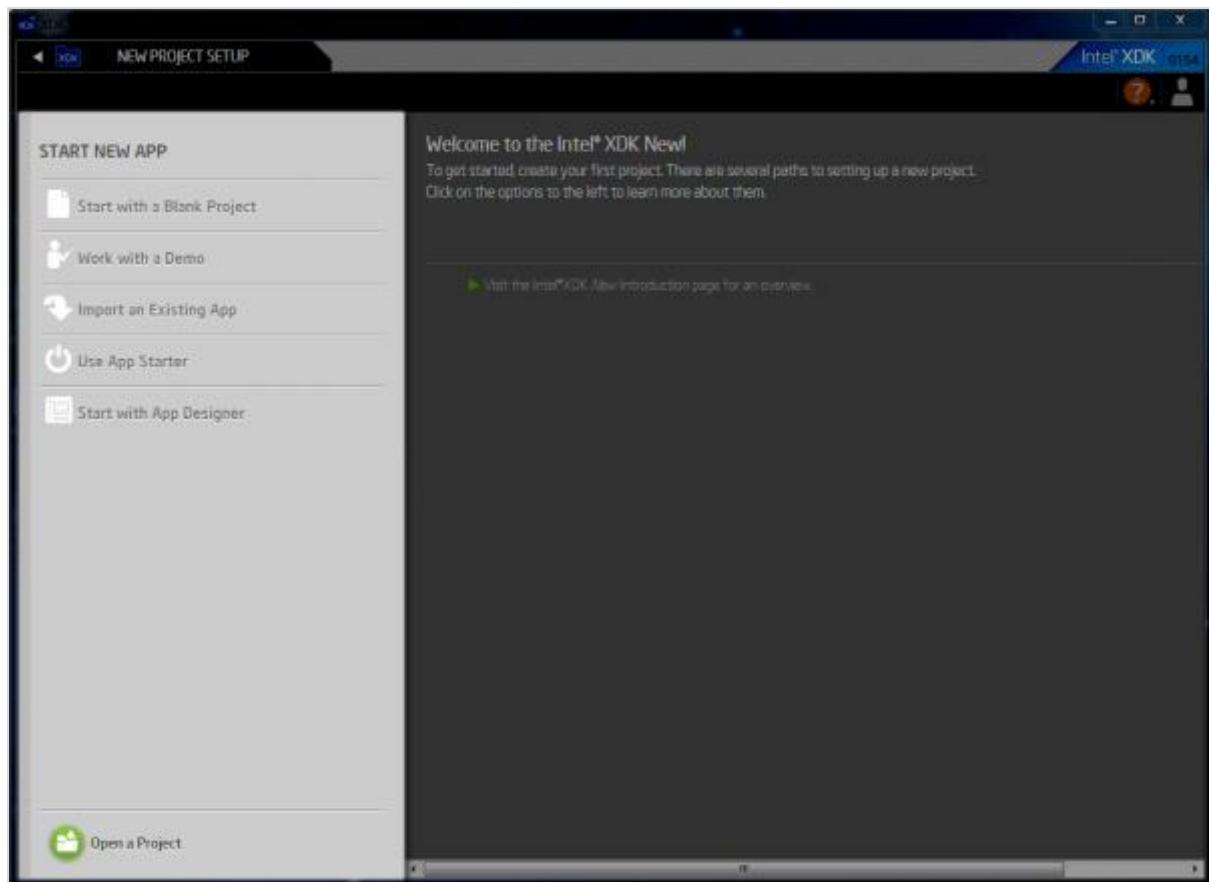
---

<sup>76</sup> Intel XDK: <https://software.intel.com/en-us/intel-xdk>

realizzazione di Web app e di applicazioni ibride per dispositivi mobili. Intel XDK è concepito per offrire tutto ciò che serve allo sviluppatore sia nella propria macchina di sviluppo sia con funzionalità remote offerte dalla potente infrastruttura server messa a disposizione da Intel.

Inoltre XDK New presenta un ambiente tipico da IDE (Integrated Development Environment) con meccanismi di supporto che mettono a proprio agio sia il programmatore che il classico designer.

Il software è disponibile per ogni sistema operativo e la procedura di installazione è la medesima. All'apertura del programma, viene presentato un menu, sulla sinistra, che mostra le varie possibilità per avviare il lavoro in XDK (figura sotto). Si può creare un nuovo progetto da zero, importarne uno precedente, aviarlo mediante una demo o partire con un progetto basato su AppStarter o AppDesigner.



Un progetto XDK è un app HTML5, senza particolari restrizioni, purché rispetti due regole:

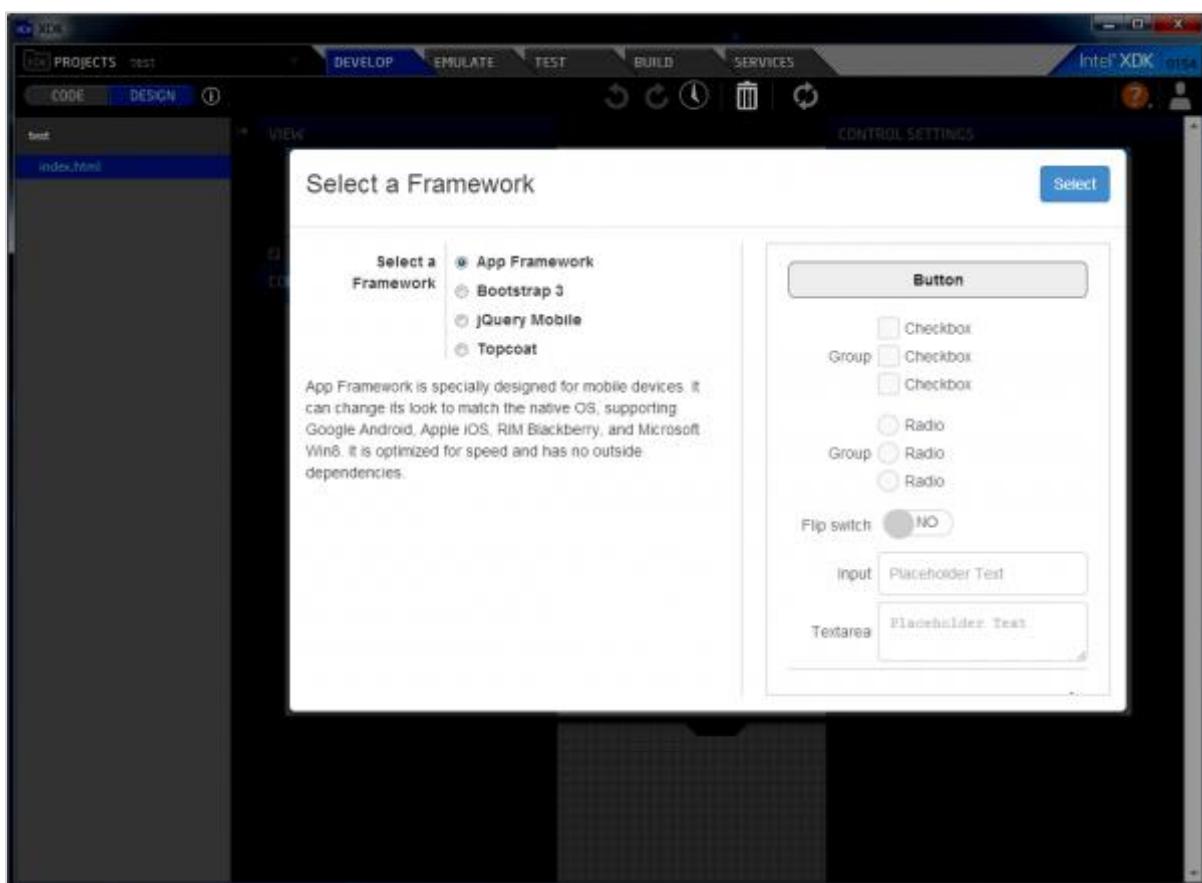
- tutti i file devono essere contenuti in una struttura gerarchica di cartelle che si dirama a partire da una cartella root;

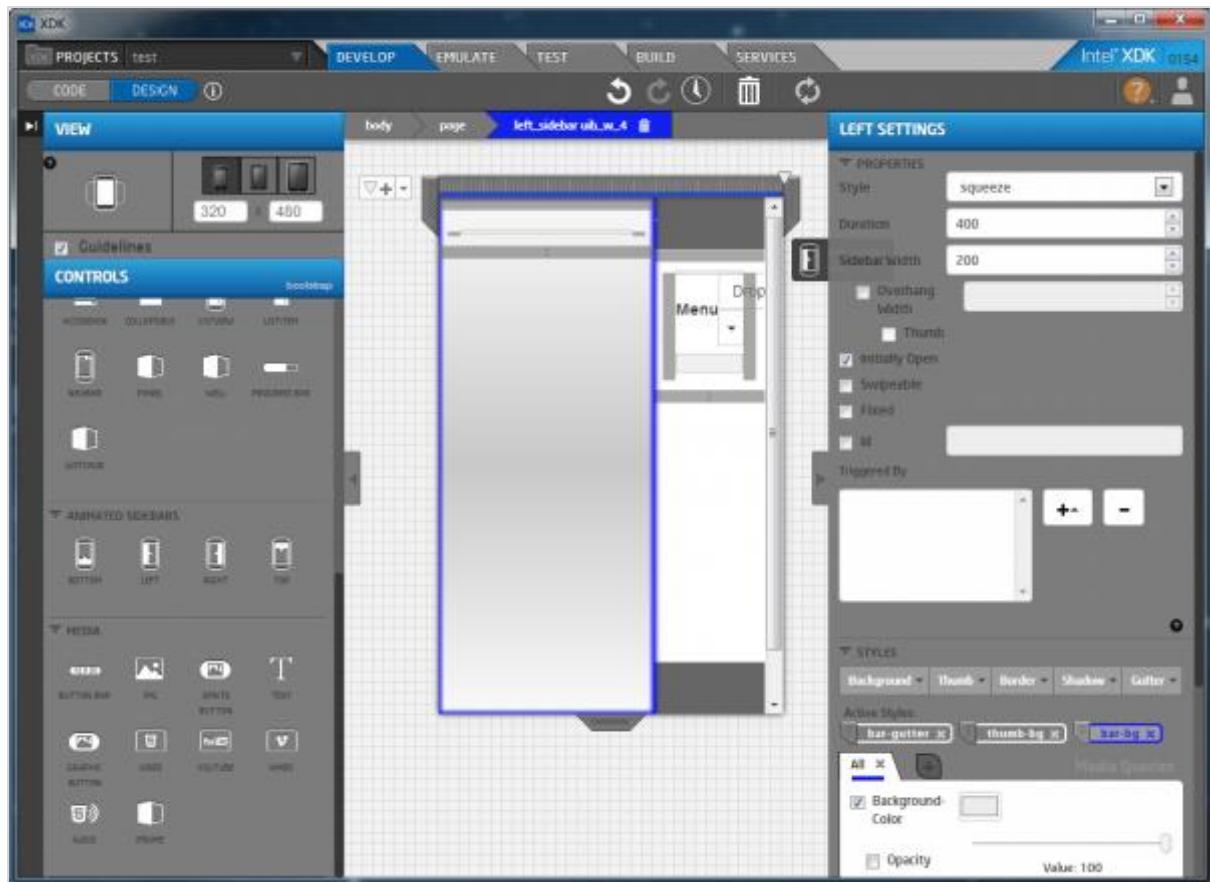
- l'applicazione prende il via da un file denominato index.html che deve essere presente nella cartella di progetto e non in sotto-cartelle.

Inoltre si consideri che una volta creato il progetto apparirà nella cartella root un file con lo stesso nome del progetto ma con estensione .xdk. Questo file non appartiene alla app ma al progetto XDK in sé stesso, ne contiene infatti la configurazione ma non è richiesto per l'esecuzione a runtime su dispositivo.

In XDK il lavoro di sviluppo si svolge con i seguenti strumenti:

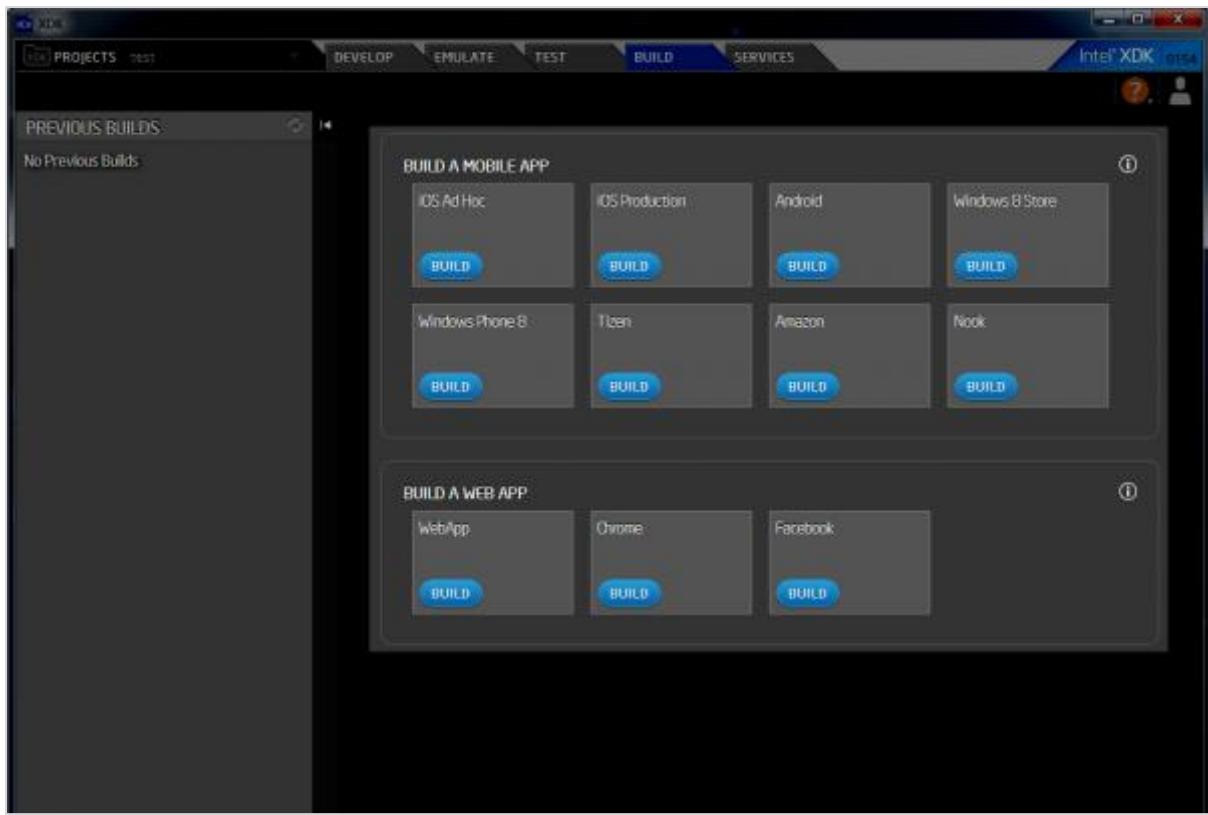
- l'editor Brackets che offre specifiche caratteristiche per la scrittura del codice HTML5, JavaScript e CSS.
- AppDesigner è un editor molto più visuale che permette di scegliere quale framework per la realizzazione delle interfacce si vuole utilizzare. Al momento di andare a definire una nuova pagina html su cui lavorare si apre una finestra dal titolo "Select a framework" che offre la possibilità di scegliere il framework preferito (figura sotto).
- App Starter è un ambiente, anch'esso visuale, che nasce per sfruttare al massimo la libreria App Framework. In particolare, offre un notevole contributo nella realizzazione di layout multi-view.





Lo sviluppatore si muoverà attraverso una sequenza di pannelli che, nel complesso, tracciano una specie di “processo produttivo”: Develop, Emulate, Test, Build e Services. In pratica ogni scheda rappresenta una fase del ciclo di produzione di un’app.

- **Develop:** sarà il terreno in cui si muoverà lo sviluppatore per costruire la propria app (figura sopra).
- **Emulate:** permette di vedere la propria app al lavoro su un emulatore che nel display riproduce la propria applicazione. Mediante un menu a tendina è possibile scegliere il dispositivo che si vuole venga emulato.
- **Test:** è finalizzato a provare l’app su un dispositivo reale. Per fare ciò è necessario che sul terminale di test sia installata un’app di nome “Intel App Preview”.
- **Build:** permette di procedere ad una compilazione vera e propria del progetto che porterà a scaricare sul proprio computer un pacchetto di installazione dell’applicazione (figura sotto).
- **Services:** raccoglie una serie di servizi di terze parti eventualmente integrabili nella propria applicazione come servizi push per la messaggistica, servizi di analytics per il tracciamento degli utilizzi dell’app, servizi di acquisto in-app e così via.



## 8.2. Login e Registrazione utente

Quella che si sta per illustrare rappresenta la prima vera user story o funzionalità che sarà implementata per dar forma al sistema. Ci si appresta ora a creare un progetto “definitivo” che sarà portato avanti nello sviluppo con incrementi successivi. Il primo momento da affrontare è quello dell'inizializzazione delle tecnologie e dei progetti.

È importante a questo punto capire di cosa si avrà bisogno ed effettuare una breve progettazione di quello che si andrà a realizzare.

La user story richiede che un qualsiasi utente che si approccia ad utilizzare il sistema sia in grado di effettuare il login per poter accedere alle funzionalità del sistema. Il login non può essere effettuato se l'utente non è memorizzato nel database, è quindi necessario considerare anche la registrazione dei dati dell'utente.

### Database

Dopo l'installazione del database MongoDB che non ha richiesto nulla in più che seguire la semplice guida<sup>77</sup>, sono state decise le caratteristiche dei dati da memorizzare.

---

<sup>77</sup> Guida installazione MongoDB: <https://docs.mongodb.com/manual/administration/install-community/>

Ogni utente sarà memorizzato come un documento nella collection “**users**” e sarà corredata dai seguenti campi:

- **\_id**: assegnato automaticamente da MongoDB;
- **name**: stringa di soli caratteri che identifica il nome di battesimo dell’utente, il campo è considerato obbligatorio;
- **surname**: stringa di soli caratteri che identifica il cognome dell’utente, il campo è considerato obbligatorio;
- **username**: stringa alfanumerica che identifica il nickname dell’utente, il campo è considerato obbligatorio;
- **password**: stringa alfanumerica che identifica la password dell’utente, il campo è considerato obbligatorio e andrà criptato prima di essere memorizzato;
- **email**: stringa in formato e-mail che identifica appunto l’indirizzo di posta elettronica dell’utente, il campo è considerato obbligatorio.

Si ricordi che essendo un database document-oriented e NoSQL, i campi possono essere aggiunti o rimossi in qualsiasi momento, pertanto il controllo sull’obbligatorietà dei campi sarà spostato sul server. Inoltre non sono necessarie relazioni di dipendenza con altre collections.

## Server NodeJS

Innanzitutto è stato necessario installare NodeJS sulla macchina; il file di installazione viene fornito con annesso NPM ossia il gestore di pacchetti dedicato a Node che contiene tutti i riferimenti ai plugin e librerie esistenti e supportate. La procedura di installazione è piuttosto semplice ed è consultabile online<sup>78</sup>.

Il passo successivo è stato quello di inizializzare il progetto tramite il comando

```
npm init
```

che si occupa di eseguire una procedura guidata per la compilazione del file “package.json” in cui sono memorizzate le informazioni di versione del progetto, dipendenze, licenza, comandi di start e di test.

---

<sup>78</sup> Guida installazione NodeJS: <https://nodejs.org/en/download/current/>

```

pepponefx@pepponefx-HP-Pavilion-g6-Notebook-PC: ~
pepponefx@pepponefx-HP-Pavilion-g6-Notebook-PC:~$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (pepponefx) █

```

Sono stati poi organizzati i file nelle cartelle nel seguente albero:

- server: cartella contenente l'intero progetto del server NodeJS.
  - app: cartella contenente tutta la logica di controllo del server.
    - models: cartella contenente gli object models necessari per controllare l'integrità dei dati inseriti.
      - *user.js: modulo JavaScript che si occupa di controllare e gestire le informazioni dell'oggetto “utente”.*
  - config: cartella contenente i file di configurazione del server.
    - *database.js: modulo contenente le variabili di accesso al database.*
    - *passport.js: modulo contenente le variabili di criptaggio della password e mantenimento delle sessioni.*
  - node\_modules: cartella creata automaticamente da Node e contenente tutte le librerie e dipendenze salvate localmente per il progetto.
  - *server.js: modulo contenente il sistema server vero e proprio, le api e la logica di business.*
  - *package.json: modulo strettamente necessario all'installazione del progetto contenente tutte le dipendenze e le informazioni sulla licenza e versione.*

Dopo la configurazione del progetto, è iniziata la fase di codifica vera e propria:

package.json

```
{
  "name": "scribaserver",
  "version": "1.0.0",
  "description": "Server per Scriba con Node JS, Express, NoSQL DB",
  "main": "server.js",
  "scripts": {
    "test": "node server.js"
  },
  "author": "Giuseppe Iaffaldano",
  "license": "MIT",
  "dependencies": {

```

```

    "bcryptjs": "^2.3.0",
    "body-parser": "^1.15.1",
    "express": "^4.14.0",
    "jwt-simple": "^0.5.0",
    "mongodb": "^2.2.1",
    "mongoose": "^4.5.0",
    "morgan": "^1.7.0",
    "node-bcrypt": "0.0.1",
    "passport": "^0.3.2",
    "passport-jwt": "^2.0.0",
    "socket.io": "^1.4.8"
  }
}

```

**bcryptjs, node-bcryptjs:** librerie per il criptaggio delle password.

**body-parser:** libreria per l'interpretazione del corpo delle richieste da parte del client.

**express**<sup>79</sup>: framework per applicazioni web Node.js flessibile e leggero che fornisce una serie di funzioni avanzate per le applicazioni web e per dispositivi mobili. Con un ricco set di metodi di utilità HTTP e middleware a disposizione, la creazione di un'API affidabile con express è un processo facile e veloce.

**jwt-simple, morgan, passport, passport-jwt:** librerie per la gestione delle sessioni.

**mongoose:** libreria ODM (Object Database Manager) per gestire gli Schemi e gli oggetti che saranno memorizzati come documenti nel database. È strettamente legato a MongoDB e fornisce metodi rapidi per la gestione dei documenti e delle collections.

database.js

```

module.exports = {
  'secret': 'STRINGA ALFANUMERICA SCELTA COME CHIAVE PER IL CRIPTAGGIO',
  'host': 'localhost',
  'port': 28015,
  'db': "scribaDB",
  'database': 'mongodb://localhost/scribaDB'
};

```

Lo script esporta il modulo contenente gli attributi dichiarati in modo da poter essere utilizzabile da altri moduli.

passport.js

```

var JwtStrategy = require('passport-jwt').Strategy;
var ExtractJwt = require('passport-jwt').ExtractJwt;

```

---

<sup>79</sup> ExpressJS: <http://expressjs.com/>

```

var User = require('../app/models/user');
var config = require('../config/database');

module.exports = function(passport) {
  var opts = {};
  opts.jwtFromRequest = ExtractJwt.fromAuthHeader();
  opts.secretOrKey = config.secret;
  passport.use(new JwtStrategy(opts, function(jwt_payload, done) {
    User.findOne({id: jwt_payload.id}, function(err, user) {
      if (err) {
        return done(err, false);
      }
      if (user) {
        done(null, user);
      } else {
        done(null, false);
      }
    });
  }));
};

```

Il modulo si occupa innanzitutto di importare le librerie necessarie, poi di importare il modello dell'utente da loggare e il file di configurazione del database, infine definisce la funzione di creazione del token univoco utilizzato per mantenere la sessione e autorizzare l'accesso a determinate API sul server.

user.js

```

var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var bcryptjs = require('bcryptjs');

// imposta un modello mongoose
var UserSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  surname: {
    type: String,
    required: true
  },
  username: {
    type: String,
    unique: true,
    required: true
  },
  password: {
    type: String,
    required: true
  },
});

```

```

email: {
    type: String,
    required: true
}
});

UserSchema.pre('save', function (next) {
    var user = this;
    if (this.isModified('password') || this.isNew) {
        bcryptjs.genSalt(10, function (err, salt) {
            if (err) {
                return next(err);
            }
            bcryptjs.hash(user.password, salt, function (err, hash) {
                if (err) {
                    return next(err);
                }
                user.password = hash;
                next();
            });
        });
    } else {
        return next();
    }
});

UserSchema.methods.comparePassword = function (passw, cb) {
    bcryptjs.compare(passw, this.password, function (err, isMatch) {
        if (err) {
            return cb(err);
        }
        cb(null, isMatch);
    });
};

module.exports = mongoose.model('User', UserSchema);

```

Il modulo definisce lo schema dell'utente con i vincoli richiesti, inoltre definisce tramite l'operazione "pre", una sequenza di istruzioni, in questo caso per il criptaggio della password, da eseguire prima della memorizzazione del documento nel database. Infine viene definito un metodo per controllare la validità della password dell'utente in fase di login.

### server.js

La parte comune a tutte le API fornite dal server e creata inizialmente come inizializzazione e pest del server è la seguente:

```

//IMPORT DELLE LIBRERIE NECESSARIE
var express = require('express');
var app = express();
var bodyParser = require('body-parser');

```

```

var morgan = require('morgan');
var mongoose = require('mongoose');
var passport = require('passport');
var config = require('./config/database');
var User = require('./app/models/user');
var port = process.env.PORT || 8080;
var jwt = require('jwt-simple');
var Event = require('./app/models/event');
var Session = require('./app/models/session');
var Intervent = require('./app/models/intervent');
var bcryptjs = require('bcryptjs');

// Add headers
app.use(function (req, res, next) {
    // Indirizzi da cui si accettano le richieste
    res.setHeader('Access-Control-Allow-Origin', '*');
    // Richieste (metodi) che si desiderano accettare
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH,
DELETE');
    // Richieste (headers) che si desiderano accettare
    res.setHeader('Access-Control-Allow-Headers', 'Authorization,X-Requested-
With,Content-Type');
    // Da impostare a true se si usano sessioni e si accettano cookie nelle richieste
    res.setHeader('Access-Control-Allow-Credentials', true);
    next();
});

// Imposta i parametri per il parsing delle richieste
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(morgan('dev'));
app.use(passport.initialize());

// Avvia il server
app.listen(port, '192.168.0.44');
console.log('Il server è in ascolto si : http://192.168.0.44:' + port);

// Connessione al database
mongoose.connect(config.database);

// Prende la chiave segreta di configurazione
require('./config/passport')(passport);

// impacchetta gli indirizzi per le API
var apiRoutes = express.Router();
app.use('/api', apiRoutes);

```

Per quanto riguarda invece la registrazione e il login degli utenti, sono state costruite le seguenti API:

```
// Crea un nuovo account utente
apiRoutes.post('/signup', function(req, res) {
  if (!req.body.username || !req.body.password || !req.body.name || !req.body.surname || !req.body.email) {
    res.json({success: false, msg: 'Passaggio di parametri incompleto'});
  } else {
    var newUser = new User({
      name: req.body.name,
      surname: req.body.surname,
      username: req.body.username,
      password: req.body.password,
      email: req.body.email
    });
    newUser.save(function(err) {
      if (err) {
        return res.json({success: false, msg: 'Username già esistente'});
      }
      res.json({success: true, msg: 'Nuovo utente creato con successo'});
    });
  }
});

// Autentica un utente generando il token
apiRoutes.post('/authenticate', function(req, res) {
  User.findOne({
    username: req.body.username
  }, function(err, user) {
    if (err) throw err;

    if (!user) {
      res.send({success: false, msg: 'Autenticazione fallita, Utente non trovato'});
    } else {
      user.comparePassword(req.body.password, function (err, isMatch) {
        if (isMatch && !err) {
          var token = jwt.encode(user, config.secret);
          res.json({success: true, token: 'JWT ' + token});
        } else {
          res.send({success: false, msg: 'Autenticazione fallita, Password non valida'});
        }
      });
    }
  });
});

// Estrae le informazioni di un utente
apiRoutes.get('/memberinfo', passport.authenticate('jwt', { session: false}),
function(req, res) {
  var token = getToken(req.headers);
  if (token) {
    var decoded = jwt.decode(token, config.secret);
```

```

User.findOne({
  username: decoded.username
}, function(err, user) {
  if (err) throw err;

  if (!user) {
    return res.send({success: false, msg: 'Autenticazione fallita, Utente non trovato'});
  } else {
    res.json({success: true, data: user});
  }
});

} else {
  return res.status(403).send({success: false, msg: 'Nessun token ricevuto'});
}
});

getToken = function (headers) {
  if (headers && headers.authorization) {
    var parted = headers.authorization.split(' ');
    if (parted.length === 2) {
      return parted[1];
    } else {
      return null;
    }
  } else {
    return null;
  }
};

```

- **/signup:** di tipo POST, richiede che tutti i campi obbligatori siano stati passati correttamente, crea un nuovo documento basato sullo schema “User” e salva il documento nel database. La chiamata al metodo “save” attiverà il “pre” definito nel file “user.js”.
- **/authenticate:** di tipo POST, richiede che username e password siano stati passati correttamente, verifica la correttezza della password usando il metodo apposito definito nel file “user.js” e in caso positivo genera il token di sessione che restituisce al chiamante.
- **/memberinfo:** di tipo GET, richiede il passaggio del token di autenticazione che viene decodificato e viene ricercato nella collection contenete gli utenti, il documento corrispondente, i dati contenuti nel documento vengono quindi restituiti al chiamante.
- **getToken(headers):** estrae dagli header della richiesta il token di autenticazione.

Il test delle API sviluppate è stato effettuato prima della scrittura del client attraverso lo strumento di casa Google “Postman<sup>80</sup>” che permette di creare richieste personalizzate e testare il corretto funzionamento del server. Per verificare invece che i dati siano stati scritti correttamente sul database è stato usato il tool gratuito “RoboAongo”. Nelle immagini seguenti è possibile vedere il test effettuato, la prima immagine di ogni test è il test dell’API e la seconda (ove presente) rappresenta la verifica di scrittura sul database.

- Test di creazione utente

The screenshot shows the Postman interface. At the top, the URL is http://192.168.0.44:8080/api/signup. The method is set to POST. The body is selected, and the type is set to x-www-form-urlencoded. The fields are: name (Giuseppe), surname (laffaldano), username (pepponefx), password (pepponefx), and email (pepponefx@gmail.com). The response status is 200 OK and the time is 268 ms. The response body is a JSON object:

```

1  {
2   "success": true,
3   "msg": "Nuovo utente creato con successo"
4 }

```

The screenshot shows the Robomongo interface. The left sidebar shows the database structure: LocalConnection (2), System, scribaDB (Collections: events, interventions, sessions, users), Functions, and Users. In the main pane, the command db.getCollection('users').find({}) is run. The results show six documents in the users collection:

Key	Value	Type
1 _id	ObjectId("57b07806c4e991283102119f")	Objectid
1 name	Giuseppe	String
1 surname	laffaldano	String
1 username	pepponefx	String
1 password	\$2a\$10\$QS9n88r2rVpVu3KGauP0TOBV...	String
1 email	pepponefx@gmail.com	String

<sup>80</sup> Postman: <https://www.getpostman.com/>

- Test di autenticazione utente

http://192.168.0.44:8080/api/authenticate

**Body**

```

1  {
2     "success": true,
3     "token": "JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
4       .eyJfaWloIi1N2iWzgNm0ZTk5MTI4MzEwMjEx0WY1LCJuYW1Ijo1R211c2VwcGuILCJzdXJuY
5       W11Ijo1SwFmZnFsZGFubyIsInVzZXJuYW1IoiGvwcG9uZW4IiwiGfz3dvcnQ101kMmEkMTAkUVMS5bjg4cjyVnBwdTNLR2F1UDBuT0JNzWF
6       WhHRS09rc3Yyv15DYBuV3NHRThvbE5kRUMiLCJlbWPpbC16InB1chHvbmvmeEBnbWFpbC5jb20iLCJFX3Y1oAsImpvaW51ZEV2Zw50cyI6W10sIm9ic2Vyd
7       m9ic2VydvmVrxZlbnRzIjpBX0.f51uPizr4M4bdA5ZP-uBxXiwe1Tw51lx1EAkazCt3SU"
8   }

```

Status: 200 OK Time: 289 ms

- Test di recupero dei dati dell'utente

http://192.168.0.44:8080/api/memberinfo

**Headers (1)**

Authorization: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWloIi1N2iWzgNm0ZTk5MTI4MzEwMjEx0WY1LCJuYW1Ijo1R211c2VwcGuILCJzdXJuY

**Body**

```

1  {
2     "success": true,
3     "data": [
4         {
5             "id": "57b07806c4e991283102119f",
6             "name": "Giuseppe",
7             "surname": "Iaffaldano",
8             "username": "pepponefx",
9             "password": "$2a$10$0S9n88r2rVpVu3KGauP0TOBVeaVluQKQ0sv2V.CaPnwsaEBULndEC",
10            "email": "pepponefx@gmail.com",
11            "v": 0,
12            "joinedEvents": [],
13            "observedEvents": []
14        }
15    ]

```

Status: 200 OK Time: 30 ms

## Server Ionic

Costruito l'impianto basilare del server e le prime API necessarie al login e alla registrazione degli utenti, il passo successivo è stato l'inizializzazione di un nuovo progetto con Ionic 2 per il server Web che gestisce gli accessi da Desktop.

Seguendo la guida<sup>81</sup> ufficiale, il procedimento è stato molto semplice, il risultato della serie di comandi eseguiti è stata una cartella così organizzata:

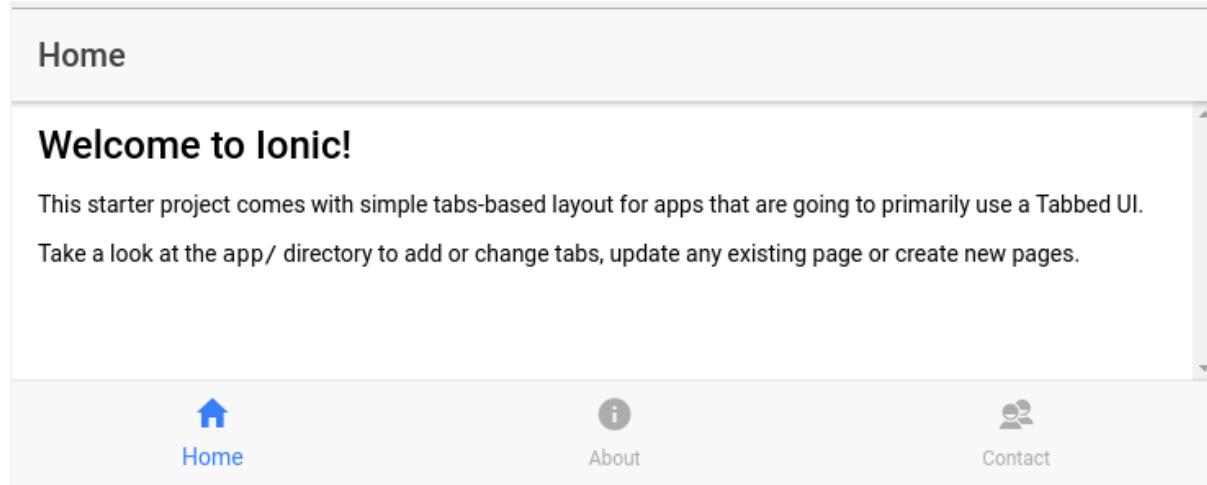
- **client:** cartella contenente l'intero progetto del client (server Ionic).
  - **app:** cartella che conterrà tutto il codice che sarà scritto per creare l'app comprendente pagine, servizi ecc.
    - **pages:** la sottocartella conterrà tutte le pagine web del sistema, ognuna con relativo controller scritto in TypeScript.
    - **services:** conterrà i servizi di interfacciamento con il server o di gestione della trascrizione e del mantenimento dei dati locali e configurazioni.
  - **hooks:** cartella contenente gli script che possono essere eseguiti come parte del processo di compilazione di Cordova. Qui vanno impostate tutte le opzioni di compilazione dei pacchetti.
  - **node\_modules:** come per il server Node, la cartella viene generata alla creazione del progetto e contiene tutti i moduli esterni importati e installati localmente per il progetto.
  - **platforms:** contiene le impostazioni, le configurazioni e le build effettuate sulle piattaforme installate, nel caso di questo progetto, iOS e Android.
  - **plugins:** contiene tutti i plugin Cordova necessari per la compilazione.
  - **resources:** contiene le immagini e le icone per le diverse piattaforme supportate.
  - **typings:** cartella contenente la definizione dei tipi per moduli esterni che non sono scritti in TypeScript.
  - **www:** questa cartella conterrà tutto il codice compilato presente in “app” oltre al file “index.html”, punto di partenza dell’intero sistema.
  - **config.xml:** contiene la configurazione per Cordova usata per la creazione dei pacchetti app nelle diverse piattaforme.
  - **gulpfile.js:** contiene le configurazioni per la compilazione del sistema e l'esecuzione come Web Service fruibile da Desktop.
  - **ionic.config.js:** contiene le configurazioni utilizzate dalla CLI Ionic quando vengono eseguiti i comandi.
  - **package.json:** come per il server, il file contiene tutte le informazioni sull'autore, la versione, la licenza e le dipendenze (librerie) necessarie all'installazione del sistema.

---

<sup>81</sup> Guida installazione Ionic 2: <http://ionicframework.com/docs/v2/getting-started/installation/>

- *tsconfig.json*: file di configurazione del compilatore TypeScript.
- *typings.json*: file dichiarativo dell’ambiente utilizzato.

Una volta inizializzato il nuovo progetto, quello che si ha a disposizione è il progetto basilare di esempio di Ionic 2, grazie al quale è possibile comprendere meglio alcune pratiche di codifica e riutilizzare del codice già scritto (immagine sotto).



Il vantaggio è stato sfruttato mantenendo come pagina iniziale quella proposta da Ionic, è stato deciso quindi di aggiungere in alto a destra i bottoni per l’accesso e la registrazione, mentre in alto a sinistra ci sarà il titolo dell’applicazione: “Scriba” (immagine sotto).



Il bottone “Registrati” permetterà di visualizzare una “modal”, ossia una scheda popup utilizzata per gli input, i form e in alcune sue varianti come avvisi (alert), in cui l’utente andrà ad inserire tutti i dati richiesti e confermare la registrazione. Alla conferma dei dati, l’app chiederà al server di registrare il nuovo utente con la chiamata POST all’API “signup”.

Il bottone “Accedi” invece, mostrerà una “modal” che richiede username e password per l’autenticazione, alla conferma dell’utente, l’applicazione chiamerà, con una POST, l’API “authenticate” sul server Node per ottenere il token di autorizzazione che l’applicazione memorizzerà per richieste successive. Al termine della procedura l’utente sarà reindirizzato sulla pagina principale della Web App.

Il codice per realizzare questi semplici passi può sembrare molto articolato a primo impatto ma risulta invece molto semplice e intuitivo.

Il progetto iniziale si presenta con un container dell’intera applicazione scritto in html (index.html) e una directory di pagine illustrate di seguito:

## tabs/tabs.html

Questa pagina rappresenta un contenitore che gestisce unicamente il menù e lo scheletro dell'interfaccia.

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Scriba
    </ion-title>
    <ion-buttons end>
      <button (click)="login()">
        <ion-icon name="log-in"> Accedi</ion-icon>
      </button>
      <button (click)="signUp()">
        <ion-icon name="person-add"> Registrati</ion-icon>
      </button>
    </ion-buttons>
  </ion-navbar>
</ion-header>

<ion-tabs>
  <ion-tab [root]="tab1Root" tabTitle="Home" tabIcon="home"></ion-tab>
  <ion-tab [root]="tab2Root" tabTitle="Informazioni" tabIcon="information-circle"></ion-tab>
  <ion-tab [root]="tab3Root" tabTitle="Contatti" tabIcon="contacts"></ion-tab>
</ion-tabs>
```

## tabs/tabs.ts

Questo script scritto in TypeScript che gestisce la logica di cambio della taba (contenuto) e di lancio delle modal per la registrazione e l'accesso.

All'inizio di ogni script viene effettuata l'importazione dei moduli necessari all'esecuzione delle funzioni implementate; segue invece la definizione della classe e il costruttore dell'oggetto oltre a tutte le funzioni implementate.

```
import {Component} from '@angular/core';
import {HomePage} from '../home-page/home-page';
import {AboutPage} from '../about-page/about-page';
import {ContactPage} from '../contact-page/contact-page';
import {SignupPage} from '../modals/signup/signup-modal';
import {LoginPage} from '../modals/login/login-modal';
import {Modal, NavController} from 'ionic-angular';

@Component({
  templateUrl: 'build/pages/tabs/tabs.html'
})
export class TabsPage {
  private tab1Root: any;
```

```

private tab2Root: any;
private tab3Root: any;
constructor(private nav: NavController) {
    // this tells the tabs component which Pages
    // should be each tab's root Page
    this.tab1Root = HomePage;
    this.tab2Root = AboutPage;
    this.tab3Root = ContactPage;
}

login(){
    let modal = Modal.create(LoginPage);
    this.nav.present(modal);
}

signUp(){
    let modal = Modal.create(SignupPage);
    this.nav.present(modal);
}
}

```

Contenuto delle tab:

Le seguenti pagine saranno implementate e presentate in sprint successivi.

- **home-page/home-page.html**: rappresenta la pagina iniziale che viene visualizzata come contenuto della tab “Home” e che presenterà (in sprint successivi) all’utente la lista di tutti gli eventi pubblici a cui può accedere previo login.
- **home-page/home-page.ts**: script che gestisce l’interazione con il servizio per recuperare gli eventi esistenti e la creazione della lista che sarà visualizzata nella pagina.
- **about-page/about-page.html**: Pagina contenente le informazioni su Scriba in forma testuale.
- **about-page/about-page.ts**: script che gestisce i contenuti variabili della pagina de informazione.
- **contact-page/contact-page.html**: pagina contenente la lista dei contatti utili del team di sviluppo e ISF.
- **contact-page/contact-page.ts**: script che gestisce la lista dei contatti da visualizzare nella pagina.

Le prime pagine implementate del progetto sono state:

modals/signup/signup-modal.html

Pagina che contiene il form con tutti i campi necessari alla registrazione di un nuovo utente e che utilizza le direttive “ng” ovvero proprie del framework Angular per gestire i controlli del form.

```

<ion-toolbar>
  <ion-title>
    Registrati
  </ion-title>
</ion-toolbar>
<form #signupForm="ngForm" (ngSubmit)="submit()">
  <ion-item>
    <ion-label floating>Nome</ion-label>
    <ion-input type="text" ngControl="nameControl" [(ngModel)]="user.name"
pattern="[a-zA-Z ]*" required></ion-input>
  </ion-item>
  <ion-item>
    <ion-label floating>Cognome</ion-label>
    <ion-input type="text" ngControl="surnameControl" [(ngModel)]="user.surname"
pattern="[a-zA-Z ]*" required></ion-input>
  </ion-item>
  <ion-item>
    <ion-label floating>Username</ion-label>
    <ion-input type="text" ngControl="usernameControl" [(ngModel)]="user.username"
minlength="4" pattern="[a-zA-Z]+[0-9]*" required></ion-input>
  </ion-item>
  <ion-item>
    <ion-label floating>Password</ion-label>
    <ion-input type="password" ngControl="passwordControl"
[(ngModel)]="user.password" minlength="8" #pwd="ngForm" required></ion-input>
  </ion-item>
  <ion-item>
    <ion-label floating>Conferma Password</ion-label>
    <ion-input type="password" ngControl="confirmPasswordControl" minlength="8"
#cpwd="ngForm" required></ion-input>
  </ion-item>
  <div *ngIf="pwd.value!=cpwd.value && cpwd.touched" class="error-box">Le password non
corrispondono</div>
  <ion-item>
    <ion-label floating>e-Mail</ion-label>
    <ion-input type="email" ngControl="emailControl" [(ngModel)]="user.email"
required></ion-input>
  </ion-item>
  <ion-item>
    <button type="submit" [disabled]="!signupForm.valid || (pwd.value!=cpwd.value &&
cpwd.touched)" class="btn btn-standard">Registrati</button>
    <button type="reset" (click)="close()">Annulla</button>
  </ion-item>
</form>

```

modals/signup/signup-modal.ts

Script che gestisce la verifica dei campi compilati sul form, crea un oggetto Utente e chiede al servizio di gestione utente di inviare ai dati da memorizzare.

```

import { Component } from '@angular/core';
import { NgForm } from '@angular/forms';
import { NavController } from 'ionic-angular';
import { UserService } from '../../../../../services/user-services';
import { User } from '../../../../../services/models/user-model';

@Component({
  templateUrl: 'build/pages/modals/signup/signup-modal.html',
  providers: [UserService]
})
export class SignupPage {
  user: User = new User("", "", "", "", "");
  constructor(private nav: NavController, private us: UserService) {}

  submit() {
    this.us.register(this.user).map(res => res.json()).subscribe(data => {
      if(data.success){
        this.nav.pop();
      }else{
        alert(data.msg);
      }
    });
  }

  close() {
    this.nav.pop();
  }
}

```

modals/login/login-modal.html

Pagina che contiene il form di accesso con username e password. Anche questa pagina utilizza per il form le direttive Angular.

```

<ion-toolbar>
  <ion-title>
    Accedi
  </ion-title>
</ion-toolbar>
<form #loginForm="ngForm" (ngSubmit)="submit()">
  <ion-item>
    <ion-label floating>Username</ion-label>
    <ion-input type="text" ngControl="usernameControl" [(ngModel)]="username"
minlength="4" pattern="[a-zA-Z]+[0-9]*" required></ion-input>
  </ion-item>
  <ion-item>
    <ion-label floating>Password</ion-label>
    <ion-input type="password" ngControl="passwordControl" [(ngModel)]="password"
minlength="8" #pwd="ngForm" required></ion-input>
  </ion-item>
  <ion-item>
    <button type="submit" [disabled]="!loginForm.valid" class="btn btn->
```

```

standard">Accedi</button>
      <button type="reset" (click)="close()">Annulla</button>
    </ion-item>
</form>

```

modals/login/login-modal.ts

Script che gestisce la verifica dei campi compilati sul form e la richiesta di autenticazione dell'utente tramite il servizio apposito descritto di seguito.

```

import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';
import { NgForm } from '@angular/forms';
import { UserService } from '../../../../../services/user-services';
import { PersonalPage } from '../../../../../personal-page/personal-page';

@Component({
  templateUrl: 'build/pages/modals/login/login-modal.html',
  providers: [UserService]
})
export class LoginPage {
  private username: string;
  private password: string;
  constructor(private nav: NavController, private us: UserService) {}

  submit(): void {
    let user = {username: this.username, password: this.password};
    this.us.login(user).map(res => res.json()).subscribe(data => {
      if(data.success){
        window.localStorage.setItem("token", data.token);
        this.us.isLoggedIn=true;
        this.nav.setRoot(PersonalPage);
      }else{
        alert(data.msg);
      }
    });
  }

  close() {
    this.nav.pop();
  }
}

```

Infine sono stati aggiunti degli script TypeScript nella cartella dei servizi per gestire le interrogazioni al server:

services/models/user-model.ts

Questo script rappresenta l'oggetto Utente e mantiene l'integrità dei dati localmente.

```

export class User {
  constructor(
    public name: string,
    public surname: string,
    public username: string,
    public password: string,
    public email: string,
  ) {}
}

```

services/user-services.ts

Conterrà tutti i metodi messi a disposizione delle pagine per gestire i dati dell'utente, dalla registrazione alla modifica degli stessi. Per il momento contiene solo la chiamata alla creazione del nuovo utente e all'autenticazione.

```

import { Http, Response, Headers, RequestOptions } from '@angular/http';
import 'rxjs/add/operator/map'
import { Observable } from 'rxjs/Observable';
import {User} from './models/user-model';
import {Injectable, Inject} from '@angular/core';

@Injectable()
export class UserService {
  private ServerWithApiUrl = "http://192.168.0.44:8080/api";
  static get parameters() {
    return [[Http]];
  }
  constructor(private http: Http, private usrData: User) {}

  login(user): Observable<Response> {
    let body = "username=" + user.username + "&password=" + user.password;
    let headers = new Headers({ 'Content-Type': ['application/x-www-form-urlencoded'] });
    let options = new RequestOptions({ headers: headers });
    return this.http.post(this.ServerWithApiUrl + '/authenticate', body, options);
  }

  register(user: User): Observable<Response> {
    let body = "name=" + user.name + "&surname=" + user.surname + "&username=" + user.username + "&password=" + user.password + "&email=" + user.email;
    let headers = new Headers({ 'Content-Type': ['application/x-www-form-urlencoded'] });
    let options = new RequestOptions({ headers: headers });
    return this.http.post(this.ServerWithApiUrl + '/signup', body, options);
  }

  getUserData(): Observable<Response>{
    let headers = new Headers({ 'Content-Type': ['application/x-www-form-urlencoded'] });
    headers.append("Authorization",window.localStorage.getItem("token"));
  }
}

```

```

        let options = new RequestOptions({ headers: headers });
        let result = this.http.get(this.ServerWithApiUrl + '/memberinfo', options);
        return result;
    }
}

```

Come è possibile notare, ogni metodo imposta una richiesta http e restituisce al chiamante un Observable. Un Observable è un oggetto che emette zero o più elementi per poi terminare con successo oppure interrompersi con un errore durante il flusso degli elementi. Dall'altra parte abbiamo gli osservatori di tipo Observer che consumano l'elemento e gestiscono la terminazione dell'Observable (oppure il suo errore). A differenza delle classiche Callback più comunemente utilizzate in JavaScript, questo pattern consente di inviare in maniera asincrona più di un risultato; il che lo rende uno strumento decisamente più potente. Tuttavia è possibile all'occorrenza trasformare un Observable in una Callback.

## Test

Il test di questa user story, come accennato precedentemente sarà organizzato in maniera da simulare il comportamento di un utente durante l'interazione con il sistema. Seppur possa essere considerato da molti un metodo grezzo e incompleto per il testing del sistema, bisogna ricordare che si tratta di una verifica preliminare e grossolana e che successivamente potranno essere effettuati test più rigorosi.

Il piano di test è previsto nella tabella seguente:

Caso d'uso: Creare profilo Utente	
<b>Descrizione</b>	L'utente crea il suo profilo personale inserendo i dati richiesti per poter successivamente accedere alle funzionalità del sistema.
<b>Pre-condizioni</b>	-
<b>Post-condizioni per Successo</b>	Il profilo utente è creato correttamente e risulta memorizzato nel database
<b>Post-condizioni per Fallimento</b>	Non viene creato nessun profilo e viene restituito all'utente un messaggio di errore esplicativo del problema
<b>Evento innescante</b>	L'utente clicca il bottone “Registrati”
<b>Attori primari</b>	Qualsiasi utente

## Scenario o (default):

1. L'utente clicca su “Registrati”

- a. Il sistema visualizza il form di registrazione
2. L'utente compila tutti i campi
  - a. Il sistema attiva il bottone "Registrati"
3. L'utente clicca il bottone "Registrati"
  - a. Il sistema crea il profilo memorizzandolo nel database e chiude la schermata di registrazione

**Scenario 1:**

2. L'utente omette uno dei campi oppure ne cancella uno dopo averlo inserito
  - a. Il sistema NON attiva il bottone "Registrati"

**Scenario 2:**

2. L'utente inserisce uno Username già utilizzato (e tutti gli altri campi richiesti)
  - a. Il sistema attiva il bottone "Registrati"
3. L'utente clicca il bottone "Registrati"
  - a. Il sistema mostra un messaggio che avvisa l'utente dello Username già in uso

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

## 9. Sprint 4 - Gestione dati: Utenti ed Eventi

Avendo constatato, durante il meeting relativo allo sprint precedente, la fattibilità del sistema con le tecnologie scelte, è stato deciso dal team di accettare i task eseguiti e definirne di nuovi per incrementare le funzionalità del sistema. In particolare, questo sprint è dedicato alle operazioni di CRUD (Create-Read-Delete-Update) degli oggetti trattati dal sistema, ovvero utenti, eventi, sessioni, interventi. Per ciascuno di questi è stata innanzitutto progettata e predisposta una collection nel database, successivamente sono state create e testate le API del server Node dedicate alla manipolazione dei dati e all'applicazione delle operazioni succitate, infine è stata predisposta l'interfaccia grazie al framework Ionic e la logica di presentazione e filtraggio delle informazioni sul lato client grazie ad Angular. In ultima istanza, come per la prima user story completata, sono stati predisposti ed effettuati dei test simulando l'interazione di un utente con il sistema.

### 9.1. Gestione Profilo Utente

Per quanto riguarda il profilo utente, le operazioni di Create e Read sono state già realizzate nello sprint precedente. Per questo motivo deve essere completato con Update e Delete che saranno descritti in questo paragrafo.

#### 9.1.1. Update

Una volta creato il proprio profilo, probabilmente l'utente avrà bisogno di modificare i propri dati o qualora ci fossero incrementi del sistema, inserirne di nuovi. Per questa esigenza, è stata predisposta la funzione di modifica dei dati personali. Si deve sottolineare però che non sarà possibile in alcun modo per l'utente modificare il proprio “Username” essendo quello che lo identifica univocamente.

##### Server Node

Le API predisposte sul lato server sono due: una relativa alla modifica del campo password che richiede una logica a sé stante; una relativa alla modifica di uno o più degli altri campi (Username escluso).

Nel file “server.js” del progetto relativo al server, sono stati quindi aggiunti i metodi che gestiscono le chiamate a tali API:

- **/updateuser:** si occupa di aggiornare i dati relativi all'utente che non siano Username e Password. In particolare, richiede il token di autenticazione dell'utente senza il quale non si potrebbe accedere all'API. Una volta decodificato il token, è

possibile dallo stesso estrarre l'Username dell'utente su cui si vuole operare. Trovato l'utente vengono modificati i campi e aggiornato il documento nel database.

- **/updatepassword:** richiede una procedura leggermente più complessa. Decodificato il token di autenticazione, come prima, viene ricercato il documento nel database relativo all'utente interessato tramite l'Username come chiave. Trovato l'utente, si procede alla verifica, tramite la libreria “bcryptjs” (di cui nei paragrafi precedenti), della vecchia password dell'utente. Constatata la reale identità dell'utente, viene modificato il campo password inserendo la nuova password criptata a sua volta tramite la medesima libreria.

Le API realizzate sono state testate tramite il tool Postman, utilizzato anche nello sprint precedente.

- Test di aggiornamento dei dati

The screenshot shows two applications used for testing and monitoring the application's database interactions.

**Postman:** A screenshot of the Postman interface. The URL is `http://192.168.0.44:8080/api/updateuser`. The method is set to `POST`. The `Body` tab is selected, showing a `x-www-form-urlencoded` payload with fields: `name` (value: `Giuseppe_updated`), `surname` (value: `Iaffaldano_updated`), `email` (value: `email.updated@gmail.com`), and `key` (value: `value`). The response status is `200 OK` and the time taken is `35 ms`. The response body is a JSON object: `{"success": true, "msg": "Aggiornamento utente eseguito"}`.

**Robomongo 0.9.0-RC9:** A screenshot of the Robomongo interface. It shows the MongoDB database structure on the left, including collections like `LocalConnection`, `System`, `scribaDB`, and `users`. On the right, the `users` collection is selected, showing a query: `db.getCollection('users').find({})`. The results table lists six documents, each representing a user. One document is expanded to show its fields: `_id` (value: `ObjectId("57b07806c4e991283102119f")`), `name` (value: `Giuseppe_updated`), `surname` (value: `Iaffaldano_updated`), `username` (value: `pepponefx`), `password` (value: `$2510SQ59n88r2rVpVu3KGauP0TOBV...`), and `email` (value: `email.updated@gmail.com`).

- Test di aggiornamento della password

The screenshot shows two applications: Postman and Robomongo.

**Postman:** A screenshot of the Postman interface. It shows a POST request to `http://192.168.0.44:8080/api/updatepassword`. The Body tab is selected, showing form-data fields: `oldPassword` (value: `pepponefx`) and `newPassword` (value: `pepponefxupdated`). The response status is `200 OK` with a time of `513 ms`. The JSON response body is:

```

1: {
2:   "success": true,
3:   "msg": "Password Aggiornata"
4: }

```

**Robomongo:** A screenshot of the Robomongo interface. It shows a database connection to `scribaDB`. In the left sidebar, under the `users` collection, there is a table showing a single document. The table has columns `Key`, `Value`, and `Type`. The document fields are:

Key	Value	Type
<code>_id</code>	<code>ObjectId("57b07806c4e991283102119f")</code>	<code>ObjectID</code>
<code>name</code>	<code>Giuseppe_updated</code>	<code>String</code>
<code>surname</code>	<code>Iaffaldano_updated</code>	<code>String</code>
<code>username</code>	<code>pepponefx</code>	<code>String</code>
<code>password</code>	<code>\$2a\$10\$181p41T6E4skx4xbRRp8MOR...</code>	<code>String</code>
<code>email</code>	<code>email.updated@gmail.com</code>	<code>String</code>

## Server Ionic

Sul lato client ovvero sul server che offre l'accesso via Web da Desktop, è stata realizzata la home page personale, dove l'utente viene reindirizzato dopo il login e una pagina (modal) di modifica dei dati. Inoltre, alla collezione di servizi legati all'utente è stato aggiunto quello per l'invio dei nuovi dati al server Node.

personal-page/personal-page.html - personal-page.ts

Questa pagina ha lo stesso ruolo di contenitore/scheletro che ricopre la pagina “tabs” descritta nello sprint precedente. In questo caso, è stato inserito un menù laterale da cui l’utente può accedere alle diverse funzionalità del sistema e un bottone che riporta nome e cognome dell’utente dal quale è possibile visualizzare e modificare i dati personali.

In questa pagina, come in molte delle successive è stato utilizzato il binding delle variabili del controller .ts all'interno della pagina html attraverso la sintassi “{{variabile}}” tipica del framework Angular.

Lo script contiene invece le variabili che vengono visualizzate nella pagina html e la logica di indirizzamento nelle varie pagine relative alle funzionalità del sistema. La struttura dello script segue quella di altri script presentati in precedenza che vede l'importazione dei moduli seguita dalla definizione della classe contenente costruttore e altri metodi.

modals/user-update/userData-modal.html - userData-modal.ts

La pagina contiene i form di modifica della password e dei dati personali separatamente. La direttiva IF fornita da Angular, permette di mostrare uno o l'altro form in base alle esigenze.

```
<ion-toolbar>
  <ion-title>
    <ion-icon name="person"> Il Mio Profilo - {{user.username}}</ion-icon> <span
*ngIf="editingPassword"> - Cambio Password</span>
  </ion-title>
</ion-toolbar>
<form *ngIf="!editingPassword" #updateUserForm="ngForm" (ngSubmit)="submit()">
  <ion-item>
    <ion-label inline>Nome</ion-label>
    <ion-input type="text" ngControl="nameControl" [(ngModel)]="user.name"
pattern="[a-zA-Z ]*" required></ion-input>
  </ion-item>
  <ion-item>
    <ion-label inline>Cognome</ion-label>
    <ion-input type="text" ngControl="surnameControl" [(ngModel)]="user.surname"
pattern="[a-zA-Z ]*" required></ion-input>
  </ion-item>
  <ion-item>
    <ion-label inline>e-Mail</ion-label>
    <ion-input type="email" ngControl="emailControl" [(ngModel)]="user.email"
required></ion-input>
  </ion-item>
  <ion-item>
    <button type="submit" [disabled]="!updateUserForm.valid" class="btn btn-
standard">Salva</button>
    <button type="reset" (click)="close()">Annulla</button>

    <button type="button" (click)="editingPassword=!editingPassword" item-
right>Cambia Password</button>
  </ion-item>
</form>

<form *ngIf="editingPassword" #passwordForm="ngForm" (ngSubmit)="submitPassword()">
  <ion-item>
    <ion-label inline>Vecchia Password</ion-label>
    <ion-input type="password" ngControl="oldPasswordControl">
```

```

[(ngModel)]="oldPassword" minlength="8" #opwd="ngForm" required></ion-input>
</ion-item>
<ion-item>
    <ion-label inline>Nuova Password</ion-label>
    <ion-input type="password" ngControl="newPasswordControl">
[(ngModel)]="newPassword" minlength="8" #npwd="ngForm" required></ion-input>
</ion-item>
<ion-item>
    <ion-label inline>Conferma Password</ion-label>
    <ion-input type="password" ngControl="confirmPasswordControl" minlength="8"
#cpwd="ngForm" required></ion-input>
</ion-item>
<div *ngIf="opwd.value==npwd.value && npwd.touched" class="error-box">Non puoi
utilizzare la vecchia Password come Nuova</div>
<div *ngIf="npwd.value!=cpwd.value && cpwd.touched" class="error-box">Le password non
corrispondono</div>
<ion-item>
    <button type="submit" class="btn btn-standard" [disabled]="!passwordForm.valid
|| (opwd.value==npwd.value &&
npwd.touched)
|| (npwd.value!=cpwd.value &&
cpwd.touched)">Salva</button>
    <button type="reset" (click)="editingPassword=!editingPassword">Annulla</button>
</ion-item>
</form>

```

Al click su determinati buttoni, è possibile modificare direttamente le variabili del controller per modificare dinamicamente lo stato della pagina. Prima dell'attivazione dei buttoni di conferma, vengono effettuati i controlli necessari sui campi inseriti in modo che non vengano inviati campi vuoti. Nel caso specifico della password invece, viene eseguito un controllo sulla nuova password che deve essere differente dalla vecchia, e uno che assicura che il campo di conferma della nuova password sia compilato correttamente.

Lo script nella stessa cartella controlla i form e implementa l'interfacciamento con il servizio di comunicazione con il server Node. Una variabile booleana "editingPassword" è decisiva per il form da presentare all'utente; viene modificata dinamicamente per permettere di modificare i dati personali o la password in momenti differenti. I metodi submit invece chiamano il servizio di modifica dei dati dell'utente.

services/user-services.ts

Al file, che rappresenta la raccolta dei servizi messi a disposizione per gestire l'utente, sono stati aggiunti i metodi per la modifica dei dati e della password che come per gli altri metodi simili, predispongono delle richieste html al server Node e viene restituito un oggetto di tipo Observable consumato (observed) dal chiamante.

## Test

È stato creato un modello di test che simula il comportamento dell'utente e che prevede determinate risposte da parte del sistema per verificare se tutto fosse stato programmato come previsto.

<b>Caso d'uso:</b> Aggiornare dati Utente	
<b>Descrizione</b>	L'utente aggiorna i dati del suo profilo personale inserendo i nuovi in un form.
<b>Pre-condizioni</b>	L'utente deve essere autenticato.
<b>Post-condizioni per Successo</b>	Il profilo utente è modificato correttamente e risulta memorizzato nel database con i nuovi dati.
<b>Post-condizioni per Fallimento</b>	Non viene modificato nessun profilo e viene restituito all'utente un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone “Nome Cognome”.
<b>Attori primari</b>	Qualsiasi utente.

### Scenario 0 (default):

1. L'utente clicca su “Nome Cognome”
  - a. Il sistema visualizza il form con i dati dell'utente evidenziando quelli modificabili
2. L'utente compila tutti i campi che desidera modificare
  - a. Il sistema attiva il bottone “Salva”
3. L'utente clicca il bottone “Salva”
  - a. Il sistema modifica il profilo memorizzandolo nel database e chiude la schermata di modifica

### Scenario 1:

2. L'utente omette uno dei campi oppure ne cancella uno
  - a. Il sistema NON attiva il bottone “Salva”

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

<b>Caso d'uso:</b> Aggiornare password Utente	
<b>Descrizione</b>	L'utente aggiorna la sua password inserendo la vecchia e la nuova in un form.
<b>Pre-condizioni</b>	L'utente deve essere autenticato.
<b>Post-condizioni per Successo</b>	Il profilo utente è modificato correttamente e risulta memorizzato nel database con i nuovi dati.
<b>Post-condizioni per Fallimento</b>	Non viene modificato nessun profilo e viene restituito all'utente un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone "Cambia Password" nella schermata dei propri dati.
<b>Attori primari</b>	Qualsiasi utente.

#### **Scenario o (default):**

1. L'utente clicca su "Cambia Password" nella schermata dei propri dati
  - a. Il sistema visualizza il form per il cambio della password
2. L'utente compila tutti i campi richiesti
  - a. Il sistema attiva il bottone "Salva"
3. L'utente clicca il bottone "Salva"
  - a. Il sistema modifica il profilo memorizzandolo nel database e chiude la schermata di modifica

#### **Scenario 1:**

2. L'utente omette uno dei campi oppure non lo compila correttamente
  - a. Il sistema NON attiva il bottone "Salva"

#### **Scenario 2:**

2. L'utente inserisce una "vecchia password" errata
  - a. Il sistema attiva il bottone "Salva"
3. L'utente clicca il bottone "Salva"
  - a. Il sistema mostra un messaggio che avvisa l'utente dell'errore

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

#### 9.1.2. Delete (Logout)

Il logout dell'utente non richiede nessuna operazione sul server. Il server Node infatti fornisce un token al momento dell'autenticazione necessario per accedere alle API. Il token

viene memorizzato localmente dal client e nel momento in cui viene eliminato, non è più possibile accedere alle API esposte dal server. Per questo, l'unica operazione da effettuare è proprio l'eliminazione di ogni traccia del token nel momento in cui l'utente sceglie di effettuare il logout dal sistema.

Se si pensa alla persistenza delle informazioni sui browser, il primo pensiero va ai cookies, ancora indispensabili per preservare dati sul PC. Negli ultimi tempi, il W3C ha inserito nella specifica HTML5 una sezione interamente dedicata al Web Storage.

La specifica introduce due meccanismi del tutto simili ai cookies per la memorizzazione di strutture dati lato client:

- **localStorage**: una chiave in questa istanza ha visibilità a livello di dominio, significa che i dati conservati nell'istanza sono visibili in tutte le finestre aperte sul dominio. Quindi questo l'oggetto va oltre la singola sessione.
- **sessionStorage**: la visibilità delle chiavi appartenenti a questa istanza, è limitata alla finestra del browser in cui questa variabile è stata creata. Una volta chiusa a finestra, i dati contenuti in questa istanza verranno distrutti assieme alla finestra stessa. Un'istanza di questo tipo è indicata per casi in cui una transazione è legata ad una singola finestra dell'utente che però può a sua volta avere aperte più finestre del browser con diverse transazioni tutte operative allo stesso tempo.

## Server Ionic

Le uniche modifiche apportate al server Ionic sono state quindi le seguenti:

personal-page/personal-page.ts

Il metodo aggiunto non fa altro che chiamare il servizio di logout e presentare la pagina iniziale.

```
logout(){
  this.us.logout();
  this.nav.setRoot(TabsPage);
}
```

services/user-services.ts

Il metodo aggiunto elimina le informazioni memorizzate localmente (utente e token). In questo modo il sistema client non sarà più in grado di accedere ai servizi del server senza effettuare nuovamente il login.

```

logout(){
    window.localStorage.clear();
}

```

## Test

Anche in questo caso è stato creato un modello di test che simula il comportamento dell'utente e che prevede determinate risposte da parte del sistema.

<b>Caso d'uso:</b> Logout Utente	
<b>Descrizione</b>	L'utente esce dal sistema.
<b>Pre-condizioni</b>	L'utente deve essere autenticato.
<b>Post-condizioni per Successo</b>	L'utente viene riportato alla Home Page.
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone “Esci”.
<b>Attori primari</b>	Qualsiasi utente.

### Scenario o (default):

1. L'utente clicca su “Esci”
  - a. Il sistema rimanda l'utente sulla home page

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

## 9.2. Gestione Eventi

L'elemento principale su cui è costruito il sistema è proprio l'evento; questa entità è pensata nelle sue diverse sfaccettature, dalla singola lezione universitaria al workshop in più giorni. In particolare si è prestata attenzione nel costruire un modello che potesse essere adatto a tutte le esigenze senza vincolare il tipo di evento di cui si parla. In generale un evento è considerato come una occasione in cui una o più persone devono esporre un argomento e diversi spettatori staranno ad ascoltare; il tutto sarà ovviamente organizzato da qualcuno che si occupa di coordinare il susseguirsi dei fatti. Si avrà quindi un oggetto composto da un titolo, una data di inizio e una data di termine per l'evento; il luogo in cui sarà svolto; il suo organizzatore e la lista delle sessioni che lo compongono. Inoltre, nel caso specifico di questo

sistema si vuole fare distinzione tra evento pubblico e privato. Nel primo caso chiunque potrà accedere ad ascoltare l'evento mentre nel secondo caso potranno partecipare solo utenti invitati o autorizzati. Per questo sprint ci si concentrerà solo sugli eventi pubblici.

### 9.2.1. Create

Spesso gli eventi sono composti da numerose sessioni, le sessioni invece, come si vedrà in seguito saranno composte da più interventi. Per questo motivo, per una migliore gestione del singolo evento e per una separazione degli elementi in grana più fine, è stato scelto di memorizzare nelle sessioni il riferimento all'intervento a cui appartiene e non mantenere nel documento dell'intervento la lista di sessioni. Anche se questo potrebbe sembrare andar contro i principi del database NoSQL, la scelta è risultata piuttosto favorevole alla manipolazione dei dati così come è stata pensata poiché ad un primo tentativo è risultato impossibile riuscire a trasformare oggetti aggregati al client in modo da poter essere ritrasformati da quest'ultimo per la manipolazione.

#### Database

Dopo le dovute considerazioni, il database è stato arricchito dalla collection “**events**” in cui ogni documento rappresentante un singolo evento avrà i seguenti campi:

- **\_id**: assegnato automaticamente dal database, identifica univocamente il documento della collection.
- **title**: stringa che rappresenta il titolo dell'evento.
- **startDate**: indica la data di inizio dell'evento nel formato ISO 8601 completo di ora.
- **endDate**: indica la data di termine dell'evento nel formato ISO 8601 completo di ora.
- **location**: stringa che indica il luogo (indirizzo) in cui si svolgerà l'evento
- **organizer**: username dell'organizzatore dell'evento
- **public**: flag booleano che rappresenta il tipo di evento; vero se è pubblico, falso se è privato. La differenza tra i due tipi è stata descritta precedentemente.
- **status**: stringa che identifica lo stato dell'evento, potrà assumere i valori “*programmed*”; “*ongoing*”; “*passed*”.

La lista delle sessioni che compongono l'evento non è riportata ma si prevede di inserire nel documento “sessione” il riferimento all'identificatore dell'evento in cui è collocata la sessione.

## Server Node

Sul lato server l'implementazione è stata piuttosto semplice, è bastato aggiungere l'API di creazione del nuovo evento dopo aver creato (come per l'Utente) lo schema mongoose del documento denominando il file “event.js”.

server.js

L'API aggiunta (**/createevent**) non fa altro che controllare che siano stati inviati nel corpo della richiesta tutti i parametri necessari alla creazione del nuovo evento, impostare i valori e salvare il nuovo documento nella collection.

Anche in questo caso le API sono state testate prima di procedere attraverso i soliti tool.

### ● Test di Creazione Evento

The screenshot displays two tools used for API testing and MongoDB management:

- Postman:** A UI for making HTTP requests. The request is a POST to `http://192.168.0.44:8080/api/createevent`. The "Body" tab shows form-data fields: title (evento di test), startDate (2016-10-15T15:30), endDate (2016-10-15T18:30), organizer (pepponefx), location (Uniba - Via Orabona, 4), status (programmed), and key (value). The response status is 200 OK with a time of 280 ms, and the JSON body is: 

```
1: { 2: "success": true, 3: "msg": "Nuovo evento creato con successo", 4: "id": "57b086bdc4e99128310211a0" 5: }
```
- Robomongo:** A MongoDB GUI. It shows a connection to "scribaDB" and the "events" collection. A query `db.getCollection('events').find({})` is run, returning the following results:

Key	Value	Type
<code>_id</code>	<code>ObjectId("57b086bdc4e99128310211a0")</code>	<code>Objectid</code>
<code>title</code>	<code>evento di test</code>	<code>String</code>
<code>startDate</code>	<code>2016-10-15 15:30:00.000Z</code>	<code>Date</code>
<code>endDate</code>	<code>2016-10-15 18:30:00.000Z</code>	<code>Date</code>
<code>location</code>	<code>Uniba - Via Orabona, 4</code>	<code>String</code>
<code>status</code>	<code>programmed</code>	<code>String</code>
<code>organizer</code>	<code>pepponefx</code>	<code>String</code>
<code>public</code>	<code>true</code>	<code>Boolean</code>

## Server Ionic

Sul lato client, ossia sul server Ionic, è stata creata invece una pagina dedicata agli eventi personali (creati dall'utente autenticato) in cui sarà possibile gestire tutti i propri eventi. In secondo luogo è stata creata la modal (contenente un form) per l'inserimento dei dati di un nuovo evento. Infine è stato creato uno script TypeScript per gestire le richieste al server Node inerenti alla gestione degli Eventi e uno script rappresentante il modello dell'oggetto Evento.

personal-events-page/personal-events-page.html

La pagina contiene lo scheletro del contenitore degli eventi che saranno visualizzati come lista. Inoltre offre un bottone per la creazione di un nuovo evento che mostrerà la modal contenente il form per inserire i dati del nuovo evento da creare.

```
<ion-header>
  <ion-navbar>
    </ion-navbar>
</ion-header>

<ion-content padding>
  <ion-header padding>
    <ion-navbar>
      <ion-title>
        Eventi Creati
      </ion-title>
      <ion-buttons end>
        <button (click)="newEvent()">
          <ion-icon name="add"> Nuovo Evento</ion-icon>
        </button>
      </ion-buttons>
    </ion-navbar>
  </ion-header>
  <!-- LISTA DEGLI EVENTI -->
</ion-content>
```

personal-events-page/personal-events-page.ts

Lo script controllerà la creazione della lista di eventi da visualizzare e il lancio della modal per la creazione del nuovo evento.

```
[Importazioni Moduli]

@Component({
  templateUrl: 'build/pages/personal-events-page/personal-events-page.html',
  providers: [EventService]
})
export class PersonalEventsPage {
```

```

private events=[];
constructor(private evts: Events, private nav: NavController, private es: EventService)
{}

ionViewWillEnter(){
    this.updateEvents();
}

updateEvents(){
    //Estraie la lista degli eventi
}

newEvent(){
    this.evts.subscribe('reloadPersonalPage',() => {
        this.updateEvents();
    });
    let modal = Modal.create(NewEventPage);
    this.nav.present(modal);
}
}

```

modal/event/event-modal.html

Questa pagina (modal) contiene il form per l'inserimento dei dati necessari alla creazione dell'evento. Prima di abilitare il bottone di conferma effettua tutti i controlli necessari sulla coerenza dei dati inseriti, in particolare delle date di inizio e termine, utilizzando i metodi dello script controller.

```

<ion-toolbar>
    <ion-title>
        Dati Nuovo Evento
    </ion-title>
</ion-toolbar>

<form (ngSubmit)="submit()" #eventForm="ngForm">
    <ion-item>
        <ion-label inline>Titolo</ion-label>
        <ion-input type="text" ngControl="titleControl" [(ngModel)]="event.title"
required></ion-input>
    </ion-item>
    <ion-item>
        <ion-label inline>Data Inizio</ion-label>
        <ion-input type="datetime-local" ngControl="startDateControl" #sd="ngForm"
[(ngModel)]="event.startDate" required></ion-input>
    </ion-item>
    <div *ngIf="sd.value<=now && sd.touched">La data di Inizio deve essere POSTERIORE a
questo momento!</div>
    <ion-item>
        <ion-label inline>Data Termine</ion-label>
        <ion-input type="datetime-local" ngControl="endDateControl" #ed="ngForm"
[(ngModel)]="event.endDate" required></ion-input>

```

```

</ion-item>
<div *ngIf="sd.value>=ed.value && ed.touched">La Data di Termine deve essere
POSTERIORE a quella di Inizio!</div>
<ion-item>
    <ion-label inline>Luogo</ion-label>
    <ion-input type="text" ngControl="locationControl" class="form-control"
[(ngModel)]="event.location" required></ion-input>
</ion-item>
<ion-item>
    <button type="submit" class="btn btn-standard" [disabled]="!eventForm.valid || 
(sd.value<=now && sd.touched) || (sd.value>=ed.value && ed.touched)">Salva</button>
    <button type="reset" (click)="close()">Annulla</button>
</ion-item>
</form>

```

modal/event/event-modal.ts

Lo script controlla il form di inserimento e contiene le variabili e i metodi necessari alla verifica dei dati e alla creazione dell'oggetto Evento.

```

[Importazioni Moduli]

@Component({
  templateUrl: 'build/pages/modals/event/event-modal.html',
  providers: [EventService]
})
export class NewEventPage {
  //RECUPERA I DATI DELL'UTENTE CORRENTE
  private localUser=JSON.parse(window.localStorage.getItem("user"));
  private user= new User(this.localUser.name, this.localUser.surname,
  this.localUser.username, this.localUser.password, this.localUser.email);
  //IMPOSTA UN NUOVO EVENTO VUOTO
  private today = new Date();
  private now = this.today.toISOString();
  private event = new Event("",this.user.username,"", this.today, this.today, "", 
"programmed");
  constructor(private np: NavParams, private evts: Events, private nav: NavController,
  private es: EventService) {}
  submit(){
    this.es.createEvent(this.event).map(res=>res.json()).subscribe(data=>{
      if(data.success){
        this.evts.publish('reloadPersonalPage');
        this.nav.pop();
      }else{
        alert(data.msg);
      }
    });
  }
  close() {
    this.nav.pop();
  }
}

```

services/models/event-model.ts

Come per l'utente, questo script rappresenta l'oggetto Evento e contiene tutti i suoi attributi, nello stesso formato utilizzato per l'oggetto Utente.

services/event-services.ts

Il file rappresenta l'equivalente dello script “user-services.ts”, che gestisce però le richieste inerenti alla gestione degli eventi. Lo script si occupa di accedere alle API del server Node per creare (successivamente saranno inserite nuove funzionalità) il nuovo evento dopo aver costruito la richiesta http.

## Test

La creazione dell'evento da parte dell'utente è stata testata simulando i possibili flussi di interazione con il sistema e verificandone il corretto comportamento.

<b>Caso d'uso:</b> Creare Evento Pubblico	
<b>Descrizione</b>	L'utente crea un nuovo evento pubblico (accessibile a tutti) inserendo le informazioni richieste.
<b>Pre-condizioni</b>	L'utente deve aver selezionato la voce “I miei Eventi” dal menù laterale.
<b>Post-condizioni per Successo</b>	L'evento viene creato e memorizzato nel database.
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone “Nuovo Evento”.
<b>Attori primari</b>	Qualsiasi utente.

### Scenario 0 (default):

1. L'utente clicca sul bottone “Nuovo Evento”.
  - a. Il sistema mostra il form di inserimento dei dati.
2. L'utente inserisce tutti i dati.
  - a. Il sistema attiva il bottone “Salva”.
3. Il sistema crea l'evento e aggiorna la pagina.

### Scenario 1

2. L'utente omette almeno un campo o inserisce un valore non valido.
  - a. Il sistema NON attiva il bottone “Salva”.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

### 9.2.2. Read

Per operazione di “Read” si intende in questo caso il recupero delle informazioni dell’evento o degli eventi dal database tramite chiamata ad una API del server Node per essere visualizzate all’utente. Gli eventi sono stati progettati per contenere informazioni utili al filtraggio come “organizer” o il flag “public” o ancora lo “status”. Questo avvantaggia notevolmente dal punto di vista delle prestazioni per il recupero dei dati e la comunicazione client-server poiché si utilizzano diverse API per accedere a diversi tipi di Evento.

#### Server Node

Sono state predisposte le API per recuperare dal database la lista degli eventi con determinate caratteristiche. In particolare si fa distinzione tra eventi pubblici e privati (API ancora da implementare) e tra gli eventi che appartengono (organizzati da) o meno ad un determinato utente. Lo script contenente le API è quindi stato aggiornato aggiungendo le seguenti.

#### server.js

```
apiRoutes.get('/personalevents', function(req, res) {
  var token = getToken(req.headers);
  if (token) {
    var decoded = jwt.decode(token, config.secret);
    Event.find({
      organizer: decoded.username
    }, function(err, eventsArray) {
      if (err) throw err;

      if (!eventsArray) {
        return res.send({success: false, msg: 'Nessun evento per questo utente trovato'});
      }
      res.json({success: true, data: eventsArray});
    });
  }else{
    return res.status(403).send({success: false, msg: 'Nessun token ricevuto'});
  }
});

apiRoutes.get('/publicevents', function(req, res) {
  Event.find({
    public: true
  }, function(err, eventsArray) {
    if (err) throw err;
```

```

if (!eventsArray) {
    return res.send({success: false, msg: 'Nessun evento pubblico trovato'});
}
res.json({success: true, data: eventsArray});
);
);

```

- **/personalevents**: chiede al database di restituire tutti e soli gli eventi creati dall'utente che ne fa richiesta.
- **/publicevents**: chiede al database di restituire tutti e soli gli eventi pubblici.

Come è possibile notare, la seconda API non richiede il token di autenticazione. La scelta è dovuta al fatto che nella pagina iniziale saranno visualizzati tutti gli eventi pubblici disponibili anche ad utenti non ancora autenticati.

Anche qui le API sono state testate come di consueto.

- Test di recupero Eventi Pubblici

The screenshot shows the Postman application interface. At the top, there is a header bar with the URL `http://192.168.0.44:8080/api`, a dropdown for 'Environment' set to 'No environment', and a 'Send' button. Below the header, the request details are shown: method 'GET', URL `http://192.168.0.44:8080/api/publicevents`, Headers tab selected, Params, Send, Save, and Generate Code buttons. Under the Headers tab, there are key-value pairs: 'key' and 'value'. The Body tab is selected, showing the response in 'Pretty' format. The response body is a JSON array of three event objects:

```

1 [ {
2   "success": true,
3   "data": [
4     {
5       "_id": "57988c6e980bb54c3b7ddb4d",
6       "title": "Presentazione",
7       "startDate": "2016-07-28T10:00:00.000Z",
8       "endDate": "2016-07-28T20:00:00.000Z",
9       "location": "Bari, Collab",
10      "status": "programmed",
11      "organizer": "peppone",
12      "public": true,
13      "v": 0,
14      "allowed": []
15    },
16    {
17      "_id": "579cc0cacale3bf413d19881",
18      "title": "Sovraposizioni",
19      "startDate": "2016-10-08T08:00:00.000Z",
20      "endDate": "2016-10-08T22:30:00.000Z",
21      "location": "Bari",
22      "status": "programmed",
23      "organizer": "peppone",
24      "public": true,
25      "v": 0,
26      "allowed": []
27    },
28    {
29      "_id": "57b086bcd4e99128310211a0",
30      "title": "evento di test",
31      "startDate": "2016-10-15T15:30:00.000Z",
32      "endDate": "2016-10-15T18:30:00.000Z",
33      "location": "Uniba - Via Orabora, 4",
34      "status": "programmed",
35      "organizer": "pepponefx",
36      "public": true,
37      "v": 0,
38      "allowed": []
39    }
40  ]
41 }

```

- Test di recupero Eventi Personalni

The screenshot shows the Postman interface with a successful API call. The URL is `http://192.168.0.44:8080/api/personalevents`. The response body is a JSON object:

```

1  [
2   {
3     "success": true,
4     "data": [
5       {
6         "id": "57b086bdc4e99128310211a0",
7         "title": "evento di test",
8         "startDate": "2016-10-15T15:30:00.000Z",
9         "endDate": "2016-10-15T18:30:00.000Z",
10        "location": "Uniba - Via Orabona, 4",
11        "status": "programmed",
12        "organizer": "pepponefx",
13        "public": true,
14        "v": 0,
15        "allowed": []
16      }
17    ]
18  ]

```

## Server Ionic

Il lato a più stretto contatto con l'utente è stato quello che ha subito le modifiche maggiori. Innanzitutto è stata implementata la lista degli interventi pubblici in home page (nessun login richiesto); dopodichè è stata inserita la lista degli eventi pubblici anche dopo aver effettuato il login nella home page personale; infine è stata aggiunta la lista degli eventi personali (creati dall'utente) nella pagina utilizzata anche per accedere alla creazione di un nuovo evento (descritta poco sopra). Modifiche minori sono state le aggiunte dei riferimenti e le importazioni delle nuove pagine negli script creati in precedenza (tali modifiche non saranno illustrate).

`home-page/home-page.html`

La pagina presenta un primo esempio del costrutto FOR messo a disposizione dal framework Angular per facilitare la scrittura di pagine web dinamiche che presentano liste di oggetti. Nel complesso la pagina mostra semplicemente la lista degli eventi pubblici disponibili con le informazioni principali.

```

<ion-content padding>
  <ion-header padding>
    <ion-navbar>
      <ion-title>

```

```

        Tutti Gli Eventi Pubblici
    </ion-title>
    </ion-navbar>
</ion-header>
<ion-card *ngFor="let event of events">
    <ion-card-header>
        <ion-toolbar>
            <ion-title>
                {{event.title}} (@{{event.location}})
            </ion-title>
            <ion-buttons end>
                <!-- Funzionalità su un singolo evento -->
            </ion-buttons>
        </ion-toolbar>
        Dal: {{event.startDate}} - Al: {{event.endDate}}
    </ion-card-header>
</ion-card>
</ion-content>

```

home-page/home-page.ts

Lo script si occupa di reperire (tramite chiamate gli appositi metodi del file “event-services.js”) la lista degli eventi pubblici disponibili e di organizzarla in una struttura leggibile dal file html tramite la direttiva FOR.

#### [Importazioni Moduli]

```

@Component({
  templateUrl: 'build/pages/home-page/home-page.html',
  providers: [EventService, UserService]
})
export class HomePage {
  private events = [];
  constructor(private evts: Events, private nav: NavController, private es: EventService,
  private us: UserService) {
    this.updateEvents();
  }

  updateEvents(){
    let _events = [];
    this.es.getPublicEvents().map(res=> res.json()).subscribe((data) => {
      data.data.forEach(event =>{
        _events.push(event);
        this.events = _events;
      });
    });
  }
}

```

personal-home-page/personal-home-page.html - personal-home-page.ts

Questa pagina non è altro che una copia quasi completa della pagina “home-page.html” dedicata agli utenti anche non autenticati. La pagina è stata duplicata poiché verranno successivamente inserite funzionalità aggiuntive per il solo utente autenticato. In un secondo momento ci si potrebbe rendere conto del fatto che questa duplicazione risulta superflua e potrebbe quindi essere eliminata senza alcun impatto sul sistema, ma semplicemente collegando più controller ad una stessa pagina che implementino comportamenti differenti. Il file script .ts controlla la creazione della lista di eventi pubblici disponibili in maniera molto simile a “home-page.ts”. La differenza principale sta nel fatto che in questo caso viene mantenuta localmente traccia dell’utente autenticato in modo da permettere diverse operazioni aggiuntive che saranno implementate successivamente. Come accennato precedentemente, la pagina html controllata da questo script potrebbe risultare un duplicato, in tal caso basterebbe sostituire il riferimento a tale pagina nella dichiarazione di questa componente.

personal-events-page/personal-events-page.html - personal-events-page.ts

La pagina precedentemente creata come scheletro, è stata completata inserendo la lista degli eventi creati dall’utente autenticato. La lista è gestita esattamente come le altre grazie all’utilizzo della direttiva FOR del framework Angular. Nella sezione precedentemente commentata è stato quindi aggiunto il seguente codice.

```
[...]
<ion-card *ngFor="let event of events">
  <ion-card-header>
    <ion-toolbar>
      <ion-title>
        {{event.title}}
      </ion-title>
      <ion-buttons end>
        <!-- Funzionalità su un singolo evento -->
      </ion-buttons>
    </ion-toolbar>
    Dal: {{event.startDate}} - Al: {{event.endDate}}
  </ion-card-header>
</ion-card>
[...]
```

Anche in questo caso è bastato compilare il metodo predisposto per il fetch degli eventi dal database. In particolare si utilizza il servizio implementato in “event-services.ts” per chiedere al server Node di estrarre dal database tutti gli eventi creati dall’utente. Il metodo nello script è stato quindi completato in questo modo:

```

updateEvents(){
  let _events = [];
  this.es.getPersonalEvents().map(res=> res.json()).subscribe((data) => {
    data.data.forEach(event =>{
      _events.push(event);
    });
    this.events = _events;
  });
}

```

## Test

Anche la lettura degli eventi da parte dell'utente è stata testata allo stesso modo per verificare che tutto sia corretto.

<b>Caso d'uso:</b> Visualizzare Eventi Pubblici	
<b>Descrizione</b>	L'utente visualizza la lista degli eventi pubblici disponibili.
<b>Pre-condizioni</b>	NESSUNA
<b>Post-condizioni per Successo</b>	L'utente visualizza nella pagina la lista degli eventi pubblici disponibili.
<b>Post-condizioni per Fallimento</b>	L'utente non visualizza nessuna lista nella pagina.
<b>Evento innescante</b>	L'utente clicca la scheda "Home" (non autenticato) oppure seleziona l'opzione "Tutti gli Eventi" dal menù laterale.
<b>Attori primari</b>	Qualsiasi utente.

### Scenario o (default):

1. L'utente clicca sulla scheda "Home" (non autenticato) oppure sull'opzione "Tutti gli Eventi" del menù laterale (autenticato).
  - a. Il sistema mostra la pagina con la lista di tutti gli eventi pubblici esistenti.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

### 9.2.3. Update

Per sua natura, un evento organizzato in una situazione reale può subire variazioni di orario, luogo, data, scaletta ecc. Per poter rispecchiare la situazione reale, un utente del sistema deve poter essere in grado di modificare (in maniera sensata) gli eventi. La decisione è stata

quindi quella di permettere agli utenti di modificare qualsiasi evento che non abbia già avuto luogo.

## Server Node

Sul lato server è bastato anche in questo caso aggiungere una API che richiedesse un identificatore dell'evento da modificare per poter recuperare dal database il documento associato e poterne modificare i campi per poi salvarlo nuovamente (**/updateevent**).

In questo caso non ci sono vincoli sulla modifica ed è possibile modificare tutti i campi eccetto l'organizzatore (che viene assegnato automaticamente).

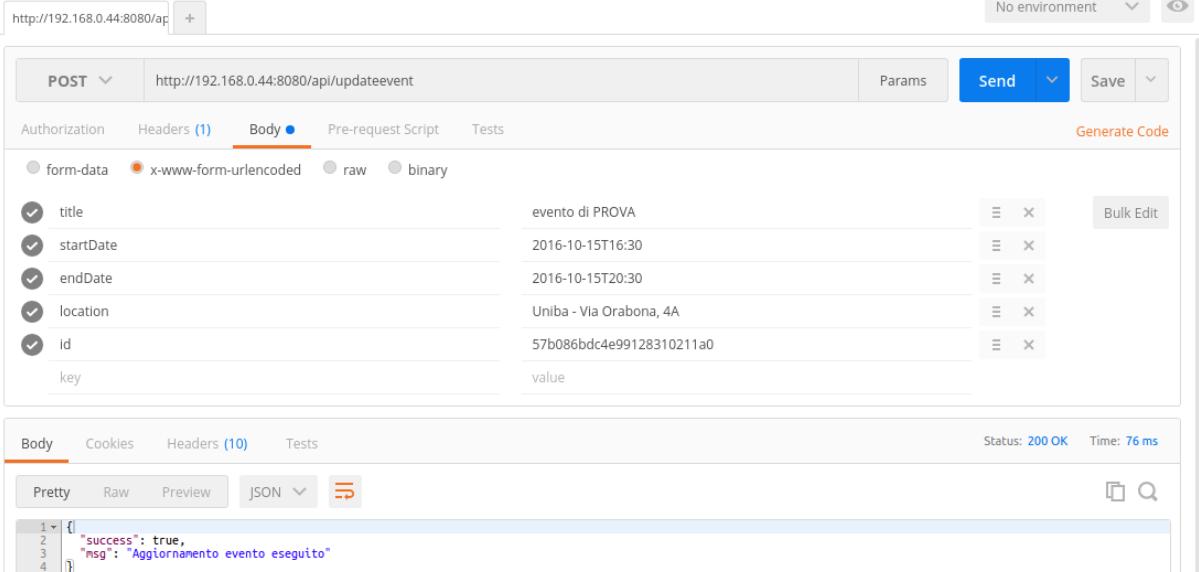
### server.js

```
apiRoutes.post('/updateevent', function(req, res) {
  var token = getToken(req.headers);
  if (token) {
    if (!req.body.title || !req.body.startDate || !req.body.endDate ||
    !req.body.location || !req.body.id) {
      res.json({success: false, msg: 'Passaggio di parametri incompleto'});
    } else {
      Event.update({
        _id: req.body.id
      }, { $set: {
        title: req.body.title,
        startDate: req.body.startDate,
        endDate: req.body.endDate,
        location: req.body.location
      }}, {multi: false}, function(err, result) {
        if (err) throw err;

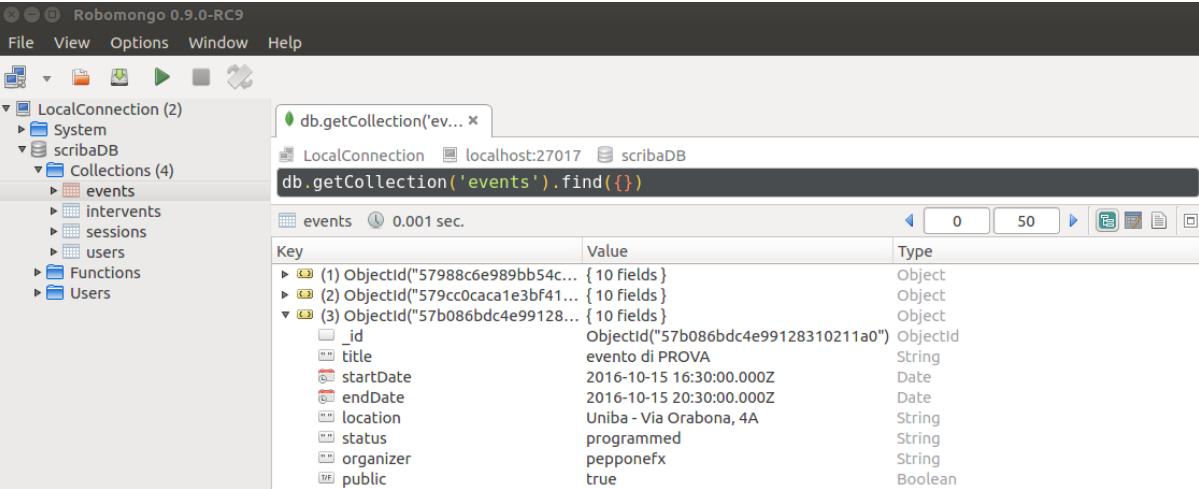
        if (!result) {
          return res.send({success: false, msg: 'Aggiornamento evento fallito'});
        } else {
          console.log(result);
          res.json({success: true, msg: "Aggiornamento evento eseguito"});
        }
      });
    }
  } else {
    return res.status(403).send({success: false, msg: 'Nessun token ricevuto'});
  }
});
```

Il test dell'API è stato effettuato anche in questo caso.

- Test di Modifica Evento



The screenshot shows the Postman interface with a POST request to `http://192.168.0.44:8080/api/updateevent`. The 'Body' tab is selected, showing form-data fields: title, startDate, endDate, location, and id, each with their respective values. The response status is 200 OK with the message "Aggiornamento evento eseguito".

The screenshot shows the Robomongo interface connected to a database named 'scribaDB'. The 'events' collection is selected. A query `db.getCollection('events').find({})` is run, and the results are displayed in a table:

Key	Value	Type
<code>_id</code>	<code>ObjectId("57b086bcd4e99128310211a0")</code>	<code>Objectid</code>
<code>title</code>	<code>evento di PROVA</code>	<code>String</code>
<code>startDate</code>	<code>2016-10-15T16:30:00.000Z</code>	<code>Date</code>
<code>endDate</code>	<code>2016-10-15T20:30:00.000Z</code>	<code>Date</code>
<code>location</code>	<code>Uniba - Via Orabona, 4A</code>	<code>String</code>
<code>status</code>	<code>programmed</code>	<code>String</code>
<code>organizer</code>	<code>pepponefx</code>	<code>String</code>
<code>public</code>	<code>true</code>	<code>Boolean</code>

## Server Ionic

Sul lato client invece, il discorso si complica leggermente poiché è stato necessario dapprima creare una pagina dedicata al singolo evento in cui vengono visualizzati tutti i dati che lo caratterizzano e che contiene i comandi (bottoni) per accedere alle funzioni di modifica dei dati; successivamente è stato aggiunto tra i “services” quello per la modifica dei dati in comunicazione con il server Node. Infine è stata aggiunto il bottone che collega alla pagina di visualizzazione del singolo evento per ciascun evento nella lista personale visualizzata all’utente.

## event-page/event-page.html

La pagina contiene semplicemente le informazioni dell'evento aperto; successivamente è stata inserita la lista delle sessioni che compongono l'evento (sarà illustrato nei paragrafi seguenti). Importante è la presenza del form “nascosto” di modifica dei dati dell'evento che compare (grazie alla direttiva IF di Angular) nel momento in cui l'utente clicca sull'icona di modifica.

```
<ion-toolbar>
</ion-toolbar>
<ion-content padding>
  <ion-header padding>
    <ion-navbar>
      <ion-title>
        {{event.title}} - Dal: {{event.startDate}} - Al: {{event.endDate}}
      </ion-title>
      <ion-buttons *ngIf="!updating && event.status=='programmed'" end>
        <button (click)="updating=!updating">
          <ion-icon name="create"> </ion-icon>
        </button>
      </ion-buttons>
    </ion-navbar>
  </ion-header>

  <form *ngIf="updating" (ngSubmit)="submit()" #eventForm="ngForm">
    <ion-item>
      <ion-label inline>Titolo</ion-label>
      <ion-input type="text" ngControl="titleControl" [(ngModel)]="newData.title" required></ion-input>
    </ion-item>
    <ion-item>
      <ion-label inline>Data Inizio</ion-label>
      <ion-input type="datetime-local" ngControl="startDateControl" #sd="ngForm" [(ngModel)]="newData.startDate"></ion-input>
    </ion-item>
    <div *ngIf="sd.value<=now && sd.touched">La data di Inizio deve essere POSTERIORE a questo momento!</div>
    <ion-item>
      <ion-label inline>Data Termine</ion-label>
      <ion-input type="datetime-local" ngControl="endDateControl" #ed="ngForm" [(ngModel)]="newData.endDate"></ion-input>
    </ion-item>
    <div *ngIf="sd.value>ed.value && ed.touched">La Data di Termine deve essere POSTERIORE a quella di Inizio!</div>
    <ion-item>
      <ion-label inline>Luogo</ion-label>
      <ion-input type="text" ngControl="locationControl" class="form-control" [(ngModel)]="newData.location" required></ion-input>
    </ion-item>
    <ion-item>
      <button type="submit" class="btn btn-standard" [disabled]="!eventForm.valid">
```

```

|| (sd.value<=now && sd.touched) || (sd.value>=ed.value && ed.touched)">Salva</button>
    <button type="reset" (click)="reset()">Annulla</button>
  </ion-item>
</form>
</ion-content>

```

event-page/event-page.ts

In questo caso lo script contiene i metodi per gestire la modifica dei dati. In particolare si interfaccia con il servizio apposito per richiedere al server Node di aggiornare l'evento aperto. Altri metodi invece sono stati aggiunti successivamente (paragrafi seguenti) per gestire la lista delle sessioni dell'evento.

```

[Importazioni Moduli]

@Component({
  templateUrl: 'build/pages/event-page/event-page.html',
  providers: [EventService]
})
export class EventPage {
  //GETS CURRENT USER
  private localUser=JSON.parse(window.localStorage.getItem("user"));
  private user= new User(this.localUser.name, this.localUser.surname,
  this.localUser.username, this.localUser.password, this.localUser.email);
  private updating = false;
  private event=this.np.get('event');
  private newData;
  private today = new Date();
  private now = this.today.toISOString();

  //SETS EVENT SESSIONS
  private sessions = [];
  constructor(private nav: NavController, private np: NavParams, private es:
  EventService, private evts: Events) {
    this.newData={_id: this.event._id, title:this.event.title, startDate:
  this.event.startDate, endDate: this.event.endDate, location: this.event.location};
  }

  ionViewWillEnter(){
    this.updateSessions(this.event._id);
  }

  updateSessions(eventID){
  //Crea la lista di sessioni dell'evento
  }

  submit(){
    if(this.newData.startDate==null){
      this.newData.startDate=this.event.startDate;
    }
    if(this.newData.endDate==null){

```

```

        this.newData.endDate=this.event.endDate;
    }
    this.es.updateEvent(this.newData).map(res=>res.json()).subscribe(data=>{
        if(data.success){
            this.nav.pop()
        }else{
            alert(data.msg)
        }
    })
}

reset(){
    this.newData={_id: this.event._id, title:this.event.title, startDate:
this.event.startDate, endDate: this.event.endDate, location: this.event.location};
    this.updating=false;
}

close() {
    this.nav.pop();
}
}

```

services/event-services.ts

È stato aggiunto in questo script il metodo per creare ed inviare la richiesta al server Node di modifica dell'evento.

```

updateEvent(event): Observable<Response>{
    let headers = new Headers({'Content-Type': ['application/x-www-form-urlencoded']});
    let body = "id=" + event._id +"&title="+ event.title + "&startDate="+
event.startDate + "&endDate=" + event.endDate + "&location=" + event.location;
    headers.append("Authorization",window.localStorage.getItem("token"));
    let options = new RequestOptions({ headers: headers });
    return this.http.post(this.ServerWithApiUrl + '/updateevent', body, options);
}

```

personal-events-page/personal-events-page.html - personal-events-page.ts

La pagina è stata arricchita con il bottone che riporta l'icona di modifica associata a ciascuna card (evento) della lista.

```

<button (click)="openEvent(event)">
    <ion-icon name="navigate"> Apri </ion-icon>
</button>

```

Così come per la pagina html associata, è stato aggiunto allo script il metodo per l'apertura della pagina relativa all'evento.

```

openEvent(eventToOpen){
    this.nav.push(EventPage, {
        event: eventToOpen,
    });
}

```

## Test

Anche in questo caso sono stati condotti dei test di simulazione del comportamento dell'utente per verificare le risposte del sistema.

<b>Caso d'uso:</b> Modificare Evento Pubblico	
<b>Descrizione</b>	L'utente modifica i dati di un evento pubblico da lui creato.
<b>Pre-condizioni</b>	L'utente deve aver aperto la pagina dell'evento da modificare; l'evento deve essere in stato “programmed”.
<b>Post-condizioni per Successo</b>	L'evento viene modificato e salvato nel database con le nuove informazioni.
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone “Modifica”.
<b>Attori primari</b>	Organizzatore dell'evento.

### Scenario 0 (default):

1. L'utente clicca sul bottone “Modifica”.
  - a. Il sistema mostra il form per la modifica dei dati.
2. L'utente modifica uno o più campi e clicca il bottone “Salva”.
  - a. Il sistema modifica le informazioni e ricarica la pagina precedente.

### Scenario 1

2. L'utente modifica uno o più campi in maniera errata
  - a. Il sistema DISATTIVA il bottone “Salva”

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

#### 9.2.4. Delete

La possibilità di eliminare un evento è una conseguenza scontata di tutto quello che è stato implementato prima. Così come nella vita reale è possibile che un evento venga annullato, anche nel sistema sarà possibile eliminare gli eventi creati.

##### Server Node

È stata semplicemente aggiunta la nuova API (**/deleteevent**) che estraе dal database l'evento interessato tramite l'identificatore richiesto al chiamante ed elimina il documento dalla collection.

##### server.js

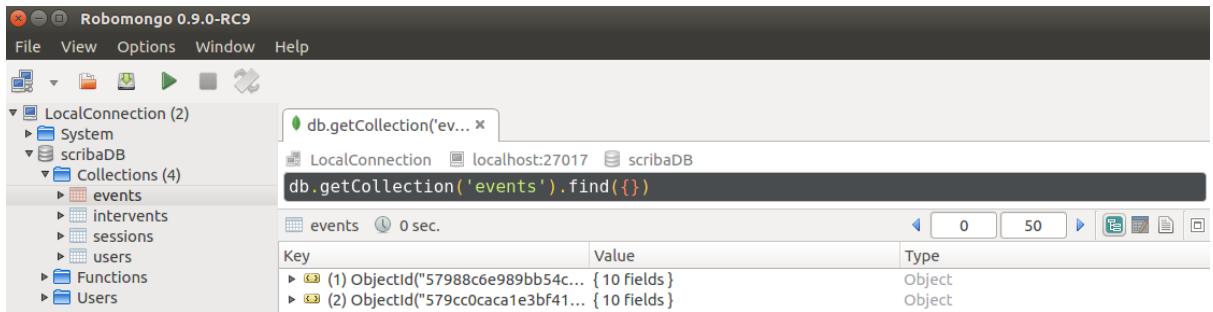
```
apiRoutes.post('/deleteevent', function(req, res) {
  var token = getToken(req.headers);
  if (token) {
    Event.remove({
      _id: req.body.id
    }, function(err, result){
      if(err) throw err;
      if (!result) {
        return res.send({success: false, msg: 'Eliminazione Evento fallita'});
      }else{
        res.json({success: true, msg: "Eliminazione Evento eseguita"});
      }
    })
  } else {
    return res.status(403).send({success: false, msg: 'Nessun token ricevuto'});
  }
});
```

Di seguito si riporta il test dell'API eseguito con i tool utilizzati precedentemente.

- Test di Eliminazione Evento

The screenshot shows the Postman application interface. At the top, the URL is set to `http://192.168.0.44:8080/api`. Below the URL bar, there are buttons for `POST`, `Send`, and `Save`. The main area shows a request configuration for a `POST` to `http://192.168.0.44:8080/api/deleteevent`. The `Body` tab is selected, showing a `x-www-form-urlencoded` key-value pair where the key is `id` and the value is `57b086bd4e99128310211a0`. Other tabs include `Authorization`, `Headers (1)`, `Params`, `Pre-request Script`, and `Tests`. A `Generate Code` button is also present. At the bottom, the response section shows a status of `200 OK` and a time of `38 ms`. The `Body` tab is selected, displaying a JSON response: 

```
1 {  
2   "success": true,  
3   "msg": "Eliminazione Evento e relative Sessioni e Interventi eseguita"  
4 }
```



## Server Ionic

È stato aggiunto il bottone iconico per ciascun evento in lista per permetterne l'eliminazione previa richiesta di conferma e il relativo metodo nello script associato. Infine è stato creato l'apposito servizio tra quelli associati alla gestione degli eventi.

personal-events-page/personal-events-page.html - personal-events-page.ts

Alla pagina è stato semplicemente aggiunto il bottone per permettere di eliminare l'evento.

```
<button (click)="deleteEvent(event)">
    <ion-icon name="trash"> </ion-icon>
</button>
```

Allo script invece è stato aggiunto il metodo che gestisce l'azione di eliminazione richiedendo al servizio apposito di eliminare l'evento passato come riferimento.

```
deleteEvent(event){
    if(event.status=="ongoing"){
        alert("Impossibile eliminare l'Evento poichè è ancora IN CORSO!")
    }else{
        let confirm = Alert.create({
            title: 'Cancellare questo Evento?',
            message: 'Se cancelli questo Evento saranno eliminati tutte le sessioni e gli interventi al suo interno e non sarà più possibile ripristinarlo!',
            buttons: [
                {
                    text: 'Cancella',
                    handler: () => {
                        this.es.deleteEvent(event._id).map(res=>res.json()).subscribe(data=>{
                            if (data.success) {
                                this.updateEvents();
                            }else{
                                alert(data.msg)
                            }
                        });
                }
            ],
        });
    }
},
```

```

        {text: 'Mantieni'}
    ];
});
this.nav.present(confirm);
}
}

```

services/event-services.ts

Il servizio predispone la richiesta al server Node inserendo header e body e restituisce il risultato come Observable al chiamante.

```

deleteEvent(eventID): Observable<Response>{
    let headers = new Headers({'Content-Type': ['application/x-www-form-urlencoded']});
    let body = "id=" + eventID;
    headers.append("Authorization",window.localStorage.getItem("token"));
    let options = new RequestOptions({ headers: headers });
    return this.http.post(this.ServerWithApiUrl + '/deleteevent', body, options);
}

```

## Test

Il comportamento dell’utente è stato simulato per verificare che il sistema rispondesse come avrebbe dovuto.

<b>Caso d’uso:</b> Cancellare Evento	
<b>Descrizione</b>	L’utente elimina un evento da lui creato e con esso tutte le sessioni e gli interventi afferenti all’evento.
<b>Pre-condizioni</b>	L’utente deve aver aperto la pagina “I Miei Eventi” dal menù laterale.
<b>Post-condizioni per Successo</b>	L’utente visualizza la lista degli eventi aggiornata (senza l’evento eliminato).
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L’utente clicca il bottone “Elimina”.
<b>Attori primari</b>	Organizzatore dell’evento.

### Scenario o (default):

1. L’utente clicca sul bottone “Elimina” di un evento.
  - a. Il sistema chiede conferma.

## 2. L'utente conferma.

- a. Il sistema elimina l'evento, le sessioni e gli interventi associati e aggiorna la pagina.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

## 9.3. Gestione Sessioni

Una volta predisposta la struttura di un evento, è giunto il momento di progettare le sessioni che lo compongono. Nei casi reali si possono individuare diversi tipi di evento, in ogni caso sono presenti in ognuno una o più sessioni, ciascuna sessione ha un argomento core su cui si sviluppa il discorso. In questo paragrafo è illustrata la struttura di una sessione e come sono state sviluppate le operazioni di CRUD su questa entità. Lo sviluppo come si vedrà è coerente con quello che è stato fatto con gli eventi.

### 9.3.1. Create

Una sessione è un momento di un evento in cui si sviluppa un certo tema con la partecipazione di uno o più esperti, pertanto è verosimile che durante una sessione intervengano più relatori per dare la propria opinione. Il moderatore od organizzatore dell'evento gestisce la scaletta e l'ordine degli interventi dei diversi relatori che possono anche alternarsi. Un aspetto rilevante è che in un evento di grossa entità, è possibile che a causa dei tempi stretti e degli argomenti molto disomogenei che si vogliono trattare, le sessioni si sovrappongano in ambienti differenti. Come di consueto, durante la fase di sviluppo delle operazioni di creazione di una sessione, è illustrata la progettazione del documento che rappresenta la stessa nel database. Sono poi chiarite le modalità di creazione dell'entità.

#### Database

Una sessione è un documento della collection “**sessions**” ed è caratterizzata dai seguenti campi:

- **\_id**: rappresenta l'identificatore del singolo documento ed è sempre assegnato automaticamente dal database.
- **title**: stringa che rappresenta il titolo della sessione, usualmente rappresenterà l'argomento su cui verte il discorso che sarà esposto.
- **startDate**: rappresenta la data di inizio della sessione comprensiva di orario nel formato ISO 8601.

- **endDate**: rappresenta la data prevista di termine della sessione nel medesimo formato ISO 8601 con orario incluso.
- **status**: stringa che rappresenta come per gli eventi se un evento è passato, programmato o in corso, assumerà quindi uno tra i valori “programmed”, “ongoing”, “passed”.
- **event**: è il riferimento all’evento in cui si colloca la sessione, sarà quindi la stringa alfanumerica identificativa del documento “evento” (`_id` dell’evento).
- **speakers**: array di stringhe che contiene gli username dei relatori che effettueranno interventi durante la sessione.

Si ricordi che i campi possono essere sempre aggiunti o rimossi o modificati successivamente.

## Server Node

Il server Node è stato arricchito con la sola API per la creazione della sessione inserita nel file “server.js” (**/createsession**) oltre che allo schema mongoose rappresentante l’oggetto inserito invece in un nuovo file “session.js”.

L’API aggiunta si occupa di controllare che siano state inviate tutte le informazioni necessarie alla creazione della sessione e di salvare un nuovo documento sessione nella collection predisposta.

Alcuni test sono stati effettuati anche in questo caso grazie ai pratici tool Postman e Robomongo. Di seguito alcuni screenshot.

- Test di Creazione della Sessione

The screenshot shows the Postman interface with a POST request to `http://192.168.0.44:8080/api/createsession`. The request body is a JSON object with the following fields:

```

{
  "title": "sessione di test",
  "startDate": "2016-10-15T16:00",
  "endDate": "2016-10-15T18:00",
  "speakers": ["pepponefx"],
  "status": "programmed",
  "event": "57b41295d0aa8c051a8e86cf"
}

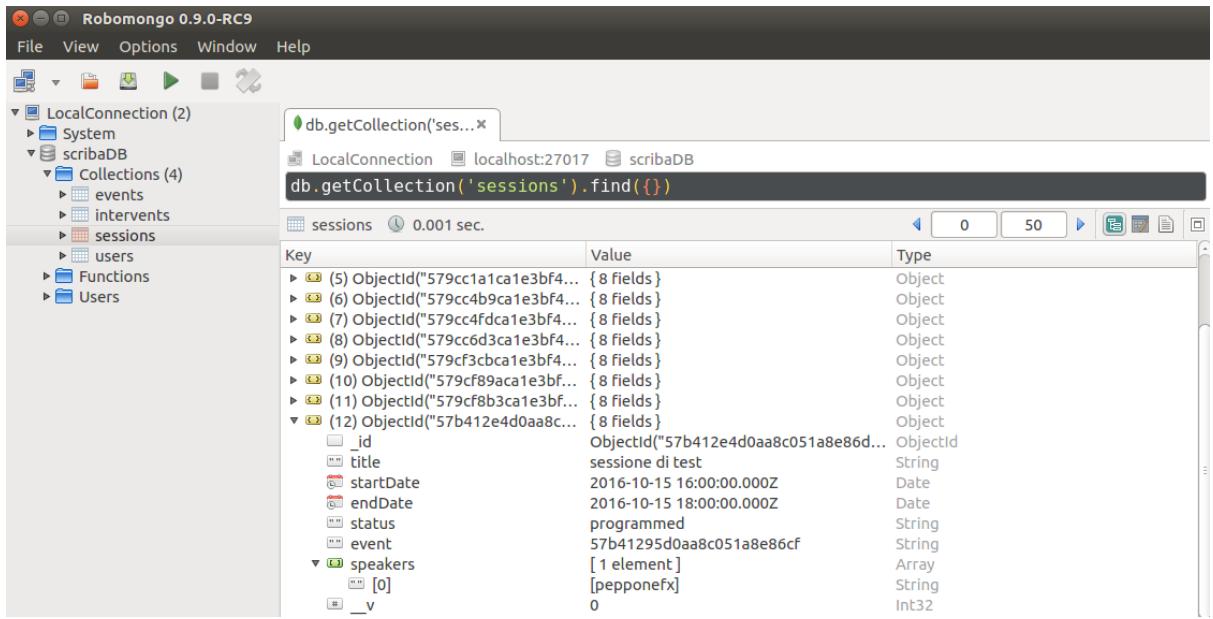
```

The response status is `200 OK` with the message:

```

{
  "success": true,
  "msg": "Nuova sessione creata con successo",
  "id": "57b412e4d0aa8c051a8e86d0"
}

```



## Server Ionic

Sul fronte dell'esposizione dei dati e funzionalità all'utente, come per gli eventi, anche in questo caso è stato creato un form modal per l'inserimento dei dati e un modello dell'oggetto sessione, accessibile dal bottone inserito accanto alla lista delle sessioni che sarà illustrata successivamente. Naturalmente anche il servizio per la comunicazione con il server Node è stato aggiunto tra quelli attinenti la gestione degli eventi nel complesso.

services/models/session-model.ts

Come per gli utenti e per gli eventi, per mantenere coerenza e controllo locale sui dati è stato predisposto l'oggetto che rappresenta la sessione con le relative informazioni.

services/event-services.ts

Ai servizi per la gestione degli eventi è stato aggiunto quello per la creazione della richiesta html da inviare al server Node per ottenere la lista delle sessioni inerenti ad un evento preciso. La lista ottenuta è restituita tramite un oggetto Observable consumato dal chiamante negli script successivi.

modals/session/session-modal.html - session-modal.ts

Il form per la creazione dell'evento è anche in questo caso una pagina a sé gestita come modal. Contiene gli appositi controlli grazie alle funzionalità fornite dal framework Angular. La particolarità dello script che controlla la pagina è che oltre ai soliti metodi per la chiusura o per l'invio dei dati, sono presenti metodi per il controllo della sovrapposizione temporale di due sessioni. In caso di sovrapposizione infatti, viene impostato un flag di overlap

(sovraposizione) che al momento dell'invio dei dati fa in modo che venga attivato il lancio di un avviso che richiede all'utente di esprimere la sua volontà di continuare con la sovrapposizione con un'altra sessione o modificare i dati per evitarla.

```
[...]
overlap(): Session{
    let overlap = null;
    let sessions=this.np.get('sessions');
    sessions.forEach(sessionInList => {
        if((sessionInList.startDate < this.session.startDate && this.session.startDate < sessionInList.endDate) ||
            (sessionInList.startDate < this.session.endDate && this.session.endDate < sessionInList.endDate) ||
            (this.session.startDate < sessionInList.startDate && sessionInList.endDate < this.session.endDate)){
            overlap = sessionInList;
        }
    });
    return overlap;
}
[...]
```

## Test

Sul lato client, il test è stato svolto simulando dei possibili comportamenti dell'utente durante l'interazione con il sistema per verificare che tutto procedesse nel verso giusto.

<b>Caso d'uso:</b> Creare Sessione Pubblica	
<b>Descrizione</b>	L'utente crea una nuova sessione per un evento pubblico (accessibile a tutti) inserendo le informazioni richieste.
<b>Pre-condizioni</b>	L'utente deve aver aperto la pagina dell'evento per cui desidera creare la sessione.
<b>Post-condizioni per Successo</b>	La sessione viene creata e memorizzata nel database.
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone "Nuova Sessione".
<b>Attori primari</b>	Organizzatore dell'evento.

## Scenario o (default):

1. L'utente clicca sul bottone "Nuova Sessione".

- a. Il sistema mostra il form di inserimento dei dati.
- 2. L'utente inserisce tutti i dati.
  - a. Il sistema attiva il bottone "Salva".
- 3. L'utente clicca il bottone "Salva".
  - a. Il sistema crea la sessione e aggiorna la pagina.

### **Scenario 1**

- 2. L'utente omette almeno un campo o inserisce un valore non valido.
  - a. Il sistema NON attiva il bottone "Salva".

### **Scenario 2**

- 2. L'utente inserisce delle date tali che la sessione risulti concomitante con un'altra.
  - a. Il sistema attiva il bottone "Salva".
- 3. L'utente clicca il bottone "Salva".
  - a. Il sistema mostra un errore e chiede se si vuole procedere o modificare i dati.
- 4. L'utente sceglie di procedere.
  - a. Il sistema crea la nuova sessione e aggiorna la pagina.

### **Scenario 3**

- 2. L'utente inserisce delle date tali che la sessione risulti concomitante con un'altra.
  - a. Il sistema attiva il bottone "Salva".
- 3. L'utente clicca il bottone "Salva".
  - a. Il sistema mostra un errore e chiede se si vuole procedere o modificare i dati.
- 4. L'utente sceglie di modificare i dati.
  - L'interazione riprende dal passo 2.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

#### **9.3.2. Read**

La lettura dei dati dal database e l'esposizione degli stessi all'utente è avvenuta come per gli eventi. In particolare, come si potrà notare, in questo caso le informazioni estratte sono state collocate in maniera coerente a far parte della sezione dedicata per ciascun evento (che mostrerà quindi la lista delle sue sessioni). Inoltre è stata creata la pagina dedicata a ciascuna sessione per poterne visualizzare le informazioni nel dettaglio e accedere alle operazioni di gestione della stessa.

## Server Node

L'API aggiunta (**/sessions**) richiede il passaggio del riferimento all'evento per il quale si vogliono estrarre le sessioni. Il sistema cercherà tutti i documenti che soddisfano le condizioni restituendo la lista al chiamante.

L'operazione è stata testata con Postman dopo aver creato un paio di sessioni di test relative all'evento precedentemente creato per il test delle API sugli eventi.

- Test di Lettura delle Sessioni

The screenshot shows a Postman interface with the following details:

- Request URL:** http://192.168.0.44:8080/api/sessions
- Method:** POST
- Body:** x-www-form-urlencoded
- Key:** event  
Value: 57b41295d0aa8c051a8e86cf

The response body is displayed in JSON format:

```
[{"id": "57b412e4d0aa8c051a8e86d0", "title": "sessione di test", "startDate": "2016-10-15T16:00:00.000Z", "endDate": "2016-10-15T18:00:00.000Z", "status": "programmed", "event": "57b41295d0aa8c051a8e86cf", "v": 0, "speakers": ["pepponefx"]}, {"id": "57b4219bd0aa8c051a8e86d1", "title": "sessione di test2", "startDate": "2016-10-15T16:00:00.000Z", "endDate": "2016-10-15T18:00:00.000Z", "status": "programmed", "event": "57b41295d0aa8c051a8e86cf", "v": 0, "speakers": ["pepponefx"]}]
```

## Server Ionic

Le sessioni di ciascun evento verranno visualizzate come preview nella lista degli eventi se l'utente decide di espandere il rispettivo menù ad albero inserito; potrà essere inoltre visualizzata all'apertura della pagina di un evento come informazione complementare dello stesso. Infine i dettagli della sessione potranno essere visualizzati nella pagina dedicata a ciascuno di essi. Di seguito vengono illustrate le modifiche effettuate e i file aggiunti.

home-page.html - personal-home-page.html - personal-events-page.html

In queste due pagine, con le medesime modalità, è stata aggiunta la lista delle sessioni in un menù ad albero in ogni scheda rappresentante un evento. Una volta espanso, l'albero mostra

la lista delle sessioni di quell'evento con le informazioni principali. Allo stesso modo saranno visualizzati gli interventi, per questo sono state predisposte le icone apposite.

```
[...]
<ion-card-content>
  <p>
    <ion-icon *ngIf="!event.expanded" name="arrow-dropdown"
(click)="event.expanded=!event.expanded"></ion-icon>
    <ion-icon *ngIf="event.expanded" name="arrow-dropup"
(click)="event.expanded=!event.expanded"></ion-icon>
    Sessioni
  </p>
  <ion-list *ngIf="event.expanded">
    <ion-item text-wrap *ngFor="let session of event.sessions">
      <p>
        <ion-icon *ngIf="!session.expanded" name="arrow-dropdown"
(click)="session.expanded=!session.expanded"></ion-icon>
        <ion-icon *ngIf="session.expanded" name="arrow-dropup"
(click)="session.expanded=!session.expanded"></ion-icon>
        {{session.title}} - {{session.startDate}}
      </p>
    </ion-item>
  </ion-list>
</ion-card-content>
[...]
```

home-page.ts - personal-home-page.ts - personal-events-page.ts

In questi tre script è stato aggiunto nel metodo che si occupa di recuperare gli eventi, una sezione di codice che si occupa invece di unire alle informazioni sull'evento, la lista delle sessioni di cui esso è composto, effettuando una chiamata all'apposito servizio esposto. Il metodo citato si trasforma quindi in quello che segue.

```
updateEvents(){
  let _events = [];
  this.es.getPublicEvents().map(res=> res.json()).subscribe((data) => {
    data.data.forEach(event =>{
      event.expanded=false;
      //TROVA E UNISCE LE SESSIONI
      let _sessions = [];
      this.es.getSessions(event._id).map(res=>res.json()).subscribe(data=>{
        data.data.forEach(session =>{
          _sessions.push(session);
          event.sessions=_sessions;})}
      _events.push(event);
      this.events = _events;
    });
  });
}
```

## event-page/event-page.html - event-page.ts

Nella pagina vengono visualizzati i dettagli di un singolo evento, di conseguenza verrà visualizzata anche la lista delle sessioni che lo compongono. Proprio come la lista di eventi vista in precedenza, quella delle sessioni avrà per ciascuna card, i comandi per gestire la sessione.

Nello script che controlla la pagina degli eventi è stato implementato il metodo per la richiesta della lista delle sessioni che lo compongono.

## services/event-services.ts

Il file è stato arricchito con il servizio per la costruzione della query all'API del server Node dedicata al recupero delle sessioni per un singolo evento.

## Test

La lettura delle sessioni è stata testata dal lato Ionic cercando di verificare se il sistema rispondesse correttamente alle richieste dell'utente.

<b>Caso d'uso:</b> Visualizzare Sessioni	
<b>Descrizione</b>	L'utente visualizza la lista delle sessioni di un evento pubblico disponibile.
<b>Pre-condizioni</b>	NESSUNA
<b>Post-condizioni per Successo</b>	L'utente visualizza nella pagina la lista delle sessioni per l'evento pubblico scelto.
<b>Post-condizioni per Fallimento</b>	L'utente non visualizza nessuna lista nella pagina.
<b>Evento innescante</b>	L'utente espande il menù ad albero per un evento nella lista.
<b>Attori primari</b>	Qualsiasi utente.

### Scenario o (default):

1. L'utente clicca sulla freccetta verso il basso del menù ad albero di un evento.
  - a. Il sistema espande l'albero mostrando la lista di tutte le sessioni dell'evento.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

### 9.3.3. Update

Anche le sessioni potrebbero subire variazioni nella scaletta e nelle date, per questo è stata predisposta la funzione di modifica dei dati accessibile dalla pagina di dettaglio della sessione con il form nascosto che viene visualizzato quando richiesto dall'utente come per gli eventi.

#### Server Node

Anche in questo caso è bastato aggiungere al server Node l'API (**/updatesession**) che riceve (dal body della richiesta html) i nuovi dati della sessione, modifica il documento interessato e lo salva nuovamente nel database aggiornandolo.

L'aggiornamento delle sessioni è stato poi testato grazie a Postman e Robomongo con la modifica di alcuni campi sulle sessioni appena create.

- Test di Modifica di una Sessione

The screenshot shows the Postman interface with a POST request to `http://192.168.0.44:8080/api/updatesession`. The 'Body' tab is selected, showing the following JSON payload:

```
[{"title": "sessione di test MODIFICATA", "startDate": "2016-10-15T16:00", "endDate": "2016-10-15T18:00", "id": "57b412e4d0aa8c051a8e86d0", "key": "value"}]
```

The response section shows a 200 OK status with the message: "success": true, "msg": "Aggiornamento sessione eseguito".

Key	Value	Type
5	ObjectId("579cc1a1ca1e3bf4...")	Object
6	ObjectId("579cc4b9ca1e3bf4...")	Object
7	ObjectId("579cc4fdca1e3bf4...")	Object
8	ObjectId("579cc6d3ca1e3bf4...")	Object
9	ObjectId("579cf3cbc1a1e3bf4...")	Object
10	ObjectId("579cf89aca1e3bf4...")	Object
11	ObjectId("579cf8b3ca1e3bf4...")	Object
12	ObjectId("57b412e4d0aa8c051a8e86d...")	Object
_id	ObjectId("57b412e4d0aa8c051a8e86d...")	Objectid
title	sessione di test MODIFICATA	String
startDate	2016-10-15 16:00:00.000Z	Date
endDate	2016-10-15 18:00:00.000Z	Date
status	programmed	String
event	57b41295d0aa8c051a8e86cf	String
speakers	[ 1 element ]	Array
__v	0	Int32

## Server Ionic

Per consentire all'utente di accedere alla funzione di modifica della sessione, è stata dapprima creata la pagina di visualizzazione dei dettagli di una singola sessione, nella stessa è stato inserito un form nascosto visualizzabile tramite l'attivazione della funzione da parte dell'utente. Infine è stato aggiunto alla lista dei servizi inerenti gli eventi, quello di modifica di una sessione.

### event-page/event-page.html - event-page.ts

Alla pagina è stato aggiunto un bottone per l'apertura dei dettagli della sessione come è stato fatto per gli eventi.

Allo script è stato aggiunto il metodo per l'apertura della pagina contenente i dettagli della sessione e le operazioni effettuabili su essa, all'apertura della pagina, come per il singolo evento, il metodo passa all'istanza della pagina dedicata alla sessione le informazioni dell'oggetto.

### session-page/session-page.html - session-page.ts

La pagina contiene il form nascosto di modifica e predisponde la visualizzazione della lista di interventi di cui sarà composta la sessione, nello stesso formato della pagina dedicata al singolo evento.

Lo script controlla che i dati inseriti siano corretti e gestisce la chiamata al servizio di modifica delle informazioni. Anche in fase di modifica viene effettuato il controllo sulla sovrapposizione della sessione che si sta modificando con le altre, in tal caso l'utente viene messo al corrente della sovrapposizione e gli viene chiesto di confermare o modificare i dati.

services/event-services.ts

Il servizio aggiunto costruisce come gli altri la richiesta al server Node restituendo il risultato allo script chiamante.

## Test

L'aggiornamento dei dati di una sessione è stato testato simulando l'operazione come se ad interagire fosse un utente reale in modo da verificare il corretto comportamento del sistema.

<b>Caso d'uso:</b> Modificare Sessione	
<b>Descrizione</b>	L'utente modifica i dati di una sessione di un evento pubblico da lui creato.
<b>Pre-condizioni</b>	L'utente deve aver aperto la pagina della sessione da modificare; la sessione deve essere in stato “programmed”.
<b>Post-condizioni per Successo</b>	La sessione viene modificata e salvata nel database con le nuove informazioni.
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone “Modifica”.
<b>Attori primari</b>	Organizzatore dell'evento.

### **Scenario o (default):**

1. L'utente clicca sul bottone “Modifica”.
  - a. Il sistema mostra il form per la modifica dei dati.
2. L'utente modifica uno o più campi e clicca il bottone “Salva”.
  - a. Il sistema modifica le informazioni e ricarica la pagina precedente.

### **Scenario 1**

2. L'utente modifica uno o più campi in maniera errata
  - a. Il sistema DISATTIVA il bottone “Salva”

### **Scenario 2**

2. L'utente modifica le date in maniera tale che la sessione risulti concomitante con un'altra e clicca su “Salva”.
  - a. Il sistema mostra un errore e chiede se si vuole procedere o modificare i dati.
3. L'utente sceglie di procedere.
  - a. Il sistema modifica i dati della sessione e ricarica la pagina precedente.

### Scenario 3

2. L'utente modifica le date in maniera tale che la sessione risulti concomitante con un'altra e clicca su "Salva".
  - a. Il sistema mostra un errore e chiede se si vuole procedere o modificare i dati.
3. L'utente sceglie di modificare i dati.
  - L'interazione riprende dal passo 2.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

#### 9.3.4. Delete

La cancellazione di una sessione è una funzione che non poteva venir meno, per questo è stata predisposta anche l'API apposita ed esposto all'utente il comando per accedervi.

##### Server Node

Aggiungendo la semplice API (**/deletesession**) alla collezione, la funzionalità sul lato server è implementata proprio come le altre. Nel caso specifico però, considerato il fatto che è stata inserita una nuova entità che dipende da un particolare oggetto evento, all'API di cancellazione dell'evento è stata aggiunta l'operazione di rimozione di tutte le sessioni ad esso associate a cascata.

È stata eliminata tramite Postman una delle due sessioni create per testare il funzionamento dell'API, in seguito è stato eliminato l'Evento a cui afferisce la rimanente per testare il corretto funzionamento della modifica effettuata sulla relativa API.

- Test di Eliminazione di una Sessione

The screenshot shows a Postman interface with the following details:

- Request URL:** http://192.168.0.44:8080/api/deletesession
- Method:** POST
- Body:** x-www-form-urlencoded
- Params:** id = 57b412e4d0aa8c051a8e86d0
- Status:** 200 OK
- Response Body (JSON):**

```
[{"success": true, "msg": "Eliminazione Sessione e relativi Interventi eseguita"}]
```

La sessione non compare più tra i documenti di MongoDB.

- Test di Eliminazione di un Evento

The screenshot shows a Postman interface with a POST request to `http://192.168.0.44:8080/api/deleteevent`. The 'Body' tab is selected, showing a single form-data entry named 'id' with the value '57b41295d0aa8c051a8e86cf'. The response tab shows a status of 200 OK with the following JSON payload:

```
{
  "success": true,
  "msg": "Eliminazione Evento e relative Sessioni e Interventi eseguita"
}
```

Né l'evento, né le sessioni ad esso associate compaiono tra i documenti del database dopo l'esecuzione.

## Server Ionic

Esporre all'utente la possibilità di eliminare una sessione è stata anche in questo caso l'operazione più semplice. È bastato aggiungere il bottone per l'eliminazione di una sessione nella card per ciascuna di esse e richiedere conferma all'utente prima di inviare (tramite lo script che controlla la pagina) la richiesta di eliminazione della sessione al servizio che si occupa di comunicare con il server Node.

`event-page/event-page.html - event-page.ts`

Alla pagina è stato aggiunto un bottone per l'eliminazione della sessione, esattamente come predisposto per l'eliminazione di un evento. Allo script è stato aggiunto il metodo per l'eliminazione della sessione che richiede conferma all'utente prima di procedere con l'azione.

`services/event-services.ts`

Il servizio è stato aggiunto agli altri nel file che gestisce i servizi relativi agli eventi comunicando con il server Node.

## Test

Come negli altri casi, è stato testato il corretto funzionamento del lato Ionic del sistema simulando l'interazione dell'utente nell'eliminazione di una sessione.

<b>Caso d'uso:</b> Cancellare Sessione	
<b>Descrizione</b>	L'utente elimina una sessione di un evento da lui creato e con essa tutti gli interventi afferenti ad essa.
<b>Pre-condizioni</b>	L'utente deve aver aperto la pagina dell'evento per cui desidera eliminare una sessione.
<b>Post-condizioni per Successo</b>	L'utente visualizza la lista delle sessioni aggiornata (senza la sessione eliminata).
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone "Elimina".
<b>Attori primari</b>	Organizzatore dell'evento.

### Scenario o (default):

1. L'utente clicca sul bottone "Elimina" di una sessione.
  - a. Il sistema chiede conferma.
2. L'utente conferma
  - a. Il sistema elimina la sessione, tutti gli interventi associati e aggiorna la pagina.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

## 9.4. Gestione Interventi

Ecco giunto il momento di parlare dell'elemento principe del sistema: l'Intervento. Questo elemento è quello che ricopre il ruolo fondamentale poiché rappresenta il punto di contatto tra relatore e partecipante di un evento. L'Intervento infatti conterrà (dinamicamente) il testo trascritto di un evento, da un lato il relatore riempirà il campo, dall'altro il partecipante riceverà la trascrizione. Anche per questo oggetto, saranno illustrate le operazioni di CRUD e ci si soffermerà maggiormente su quella di Read essendo l'operazione che ci si aspetta più frequente sulla quale saranno effettuati i test di prestazione più rigorosi.

#### 9.4.1. Create

La creazione di un intervento come struttura basilare è vista come le sessioni e gli eventi, ovvero sarà presente un documento nel database per ciascun intervento associato tramite un riferimento ad una sessione di cui fa parte e ad un relatore (Utente) responsabile dello stesso.

Durante un evento, come è stato illustrato, possono essere programmate più sessioni, durante ogni sessione possono intervenire uno o più relatori, ciascuno di essi, si dice, effettua un Intervento. In altre parole espone le sue argomentazioni e tutto ciò che viene raccontato e trascritto passa attraverso l'oggetto Intervento per poter essere fruito dall'altra parte (partecipante).

#### Database

Dal punto di vista della persistenza, un Intervento è un elemento caratterizzato da un titolo, un relatore responsabile, una data di inizio ed una durata. Inoltre, è risultato opportuno mantenere anche il testo trascritto come campo del documento, oltre ad altri flag utilizzati in maniera trasparente per la parte tecnica.

Nel database un Intervento è un documento della collection “**intervents**” contenente i seguenti campi:

- **\_id**: è il consueto identificatore del singolo documento assegnato in maniera automatica da MongoDB.
- **title**: stringa che rappresenta il titolo dell'intervento, di solito fulcro di ciò che verrà illustrato durante il suo svolgimento.
- **date**: data in formato ISO 8601 completa di ora che rappresenta l'inizio previsto dell'intervento.
- **duration**: durata stimata in minuti dell'intervento.
- **speaker**: stringa che rappresenta l'username del relatore che tiene l'intervento.
- **session**: riferimento (objectID) al documento sessione a cui afferisce l'intervento.
- **status**: stringa flag che rappresenta lo stato istantaneo dell'intervento; può assumere valore “programmed”, “ongoing” o “passed”.
- **port**: inizialmente vuoto, questo campo sarà riempito al momento dell'avvio dell'intervento con il numero della porta TCP su cui avviene la comunicazione tra relatore e partecipante.
- **text**: rappresenta il testo trascritto dell'intervento, ossia tutto quello che viene proferito dal relatore e trascritto dal sistema durante lo svolgimento dell'intervento.

Senza intaccare l'integrità dei dati, grazie alla struttura di MongoDB sarà possibile eventualmente aggiungere, rimuovere o modificare alcuni campi dei documenti anche in maniera parziale (non su tutti i documenti).

## Server Node

Sul server è bastato aggiungere ai modelli (schemi) quello che rappresenta un Intervento, tra le API è stata invece aggiunta quella per la creazione di un nuovo intervento (**/createintervent**). Quest'ultima controlla come al solito che tutti i parametri necessari siano stati passati e crea il documento inserendolo nella collection del database. Si può notare che oltre alle consuete operazioni, viene effettuata un'operazione di aggiornamento del documento rappresentante la sessione a cui afferisce l'intervento, inserendo, qualora non fosse già presente, l'username del relatore dell'intervento che si sta creando, nella lista di speaker della sessione.

```
apiRoutes.post('/createintervent', function(req, res) {
  var token = getToken(req.headers);
  if (token) {
    if (!req.body.title || !req.body.date || !req.body.duration || !req.body.speaker || !req.body.session || !req.body.status) {
      res.json({success: false, msg: 'Passaggio di parametri incompleto'});
    } else {
      Session.findOne({
        _id: req.body.session
      }, function(err, session) {
        if(err){
          return res.json({success: false, msg: 'Errore di creazione Intervento, Nessuna Sessione trovata con ID ricevuto'});
        }
        var newIntervent = new Intervent({
          title: req.body.title,
          date: req.body.date,
          duration: req.body.duration,
          speaker: req.body.speaker,
          session: req.body.session,
          status: req.body.status
        });
        newIntervent.save(function(err) {
          if (err) {
            return res.json({success: false, msg: 'Errore di creazione Intervento'});
          }
          //SE NON ESISTE GIA' UNO SPEAKER CON QUEL NOME
          if(session.speakers.indexOf(req.body.speaker) == -1){
            //AGGIORNA GLI SPEAKER DELLA SESSIONE
            Session.update({
              _id: req.body.session
            }, { $push: {
```

```

        speakers: req.body.speaker
    }, {multi: false}, function(err, result) {
        if (err) throw err;

        if (!result) {
            return res.json({success: false, msg: 'Errore di
aggiornamento Sessione'});
        }
    });
}
res.json({success: true, msg: 'Nuovo intervento creato con successo',
id: newIntervent._id});
);
}
} else{
    return res.status(403).send({success: false, msg: 'Nessun token ricevuto'});
}
});

```

La creazione dell'intervento è stata testata mediante Postman e Robomongo per verificarne il corretto funzionamento.

- Test di Creazione di un Intervento

The screenshot shows the Postman interface with the following details:

- Request URL:** http://192.168.0.44:8080/api/createintervent
- Method:** POST
- Body (x-www-form-urlencoded):**

title	Terzo intervento di test
date	2016-10-15T16:00
duration	60
speaker	speaker3
status	programmed
session	57b423edd0aa8c051a8e86d3
key	value
- Response:**
  - Status: 200 OK
  - Time: 44 ms
  - Body (Pretty):

```

1 * [{"success": true,
2   "msg": "Nuovo intervento creato con successo",
3   "id": "57b4248dd0aa8c051a8e86d3"}]
4
5

```

Key	Value	Type
» (1) ObjectId("57988cc6989bb54c...")	{ 10 fields }	Object
» (2) ObjectId("57988cd989bb54c...")	{ 8 fields }	Object
» (3) ObjectId("57b42460d0aa8c05...")	{ 8 fields }	Object
□ _id	ObjectId("57b42460d0aa8c051a8e86d5")	ObjectId
□ title	Primo intervento di test	String
□ date	2016-10-15 16:00:00.000Z	Date
□ duration	60	Int32
□ speaker	speaker1	String
□ session	57b423edd0aa8c051a8e86d3	String
□ status	programmed	String
□ __v	0	Int32
» (4) ObjectId("57b42470d0aa8c05...")	{ 8 fields }	Object
□ _id	ObjectId("57b42470d0aa8c051a8e86d7")	ObjectId
□ title	Terzo intervento di test	String
□ date	2016-10-15 16:00:00.000Z	Date
□ duration	60	Int32
□ speaker	speaker3	String
□ session	57b423edd0aa8c051a8e86d3	String
□ status	programmed	String
□ __v	0	Int32
» (5) ObjectId("57b42481d0aa8c05...")	{ 8 fields }	Object
□ _id	ObjectId("57b42481d0aa8c051a8e86d8")	ObjectId
□ title	Secondo intervento di test	String
□ date	2016-10-15 16:00:00.000Z	Date
□ duration	60	Int32
□ speaker	speaker2	String
□ session	57b423edd0aa8c051a8e86d3	String
□ status	programmed	String
□ __v	0	Int32

- Verifica di Aggiornamento della Sessione

Key	Value	Type
» (1) ObjectId("57988c91989bb54c...")	{ 8 fields }	Object
» (2) ObjectId("57988ca989bb54c...")	{ 8 fields }	Object
» (3) ObjectId("579cc0fdca1e3bf41...")	{ 8 fields }	Object
» (4) ObjectId("579cc11aca1e3bf4...")	{ 8 fields }	Object
» (5) ObjectId("579cc11aca1e3bf4...")	{ 8 fields }	Object
» (6) ObjectId("579cc4b9ca1e3bf4...")	{ 8 fields }	Object
» (7) ObjectId("579cc4fdca1e3bf4...")	{ 8 fields }	Object
» (8) ObjectId("579cc6d3ca1e3bf4...")	{ 8 fields }	Object
» (9) ObjectId("579cf3cbc1e3bf41...")	{ 8 fields }	Object
» (10) ObjectId("579cf89aca1e3bf4...")	{ 8 fields }	Object
» (11) ObjectId("579cf8b3ca1e3bf...")	{ 8 fields }	Object
» (12) ObjectId("57b423edd0aa8c0...")	{ 8 fields }	Object
□ _id	ObjectId("57b423edd0aa8c051a8e86d3")	ObjectId
□ title	sessione di test	String
□ startDate	2016-10-15 16:00:00.000Z	Date
□ endDate	2016-10-15 18:00:00.000Z	Date
□ status	programmed	String
□ event	57b41295d0aa8c051a8e86cf	String
□ speakers	[ 4 elements ]	Array
□ [0]	[pepponefx]	String
□ [1]	speaker1	String
□ [2]	speaker3	String
□ [3]	speaker2	String
□ __v	0	Int32
» (13) ObjectId("57b423f5d0aa8c0...")	{ 8 fields }	Object

Come è possibile notare, oltre allo speaker organizzatore, sono stati aggiunti gli altri tre speaker per i tre interventi creati.

### Server Ionic

Sul lato Ionic, è stata finalmente completata la collezione di oggetti locali che rappresentano le entità in gioco; è stato inserito il servizio di gestione di chiamate al server Node, per creare un intervento; creato il form modal per l'inserimento dei dati da parte dell'utente, con relativo script di controllo dati; completate le funzioni lasciate in sospeso nei file html e ts della singola sessione. Per permettere l'accesso alla funzione è stato aggiunto il bottone nella pagina della singola sessione accanto all'intestazione della lista degli interventi (codice leggibile nella sezione dedicata all'operazione Read).

#### services/models/intervent-model.ts

Rappresenta il modello locale di un Intervento ed è necessario per mantenere coerenza nei dati e controllo delle informazioni come per il modello della singola sessione od il singolo evento.

#### services/event-services.ts

È stata aggiunto il metodo per la costruzione della richiesta di creazione del nuovo intervento destinata al server Node, viene restituito al chiamante l'oggetto Observable contenente il risultato della chiamata.

#### modals/intervent/intervent-modal.html - intervent-modal.ts

Form di acquisizione delle informazioni sul nuovo intervento che controlla, grazie alle funzioni esposte dal framework Angular, la correttezza dei dati abilitando in tal caso il bottone di conferma.

Lo script è molto simile a quello dedicato alla pagina di creazione di una nuova sessione, in questo caso però, considerato il fatto che in una situazione reale, due interventi non possono sovrapporsi durante una sessione, il controllo effettuato sull'overlappe (sovraposizione) è decisivo per la possibilità di proseguire o meno nella creazione dell'intervento.

## Test

Esattamente come per le altre funzioni, il comportamento del sistema è stato verificato attraverso la simulazione delle possibili interazioni di un utente con esso.

<b>Caso d'uso:</b> Creare Intervento Pubblico	
<b>Descrizione</b>	L'utente crea un nuovo intervento per una sessione di un evento pubblico (accessibile a tutti) inserendo le informazioni richieste.
<b>Pre-condizioni</b>	L'utente deve aver aperto la pagina della sessione per cui desidera creare l'intervento.
<b>Post-condizioni per Successo</b>	L'intervento viene creato e memorizzato nel database.
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone “Nuovo Intervento”.
<b>Attori primari</b>	Organizzatore dell'evento.

### **Scenario 0 (default):**

1. L'utente clicca sul bottone “Nuovo Intervento”.
  - a. Il sistema mostra il form di inserimento dei dati.
2. L'utente inserisce tutti i dati.
  - a. Il sistema attiva il bottone “Salva”.
3. L'utente clicca il bottone “Salva”.
  - a. Il sistema crea l'intervento e aggiorna la pagina.

### **Scenario 1**

2. L'utente omette almeno un campo o inserisce un valore non valido.
  - a. Il sistema NON attiva il bottone “Salva”.

### **Scenario 2**

2. L'utente inserisce una data e una durata tali che l'intervento risulti concomitante con un altro.
    - a. Il sistema attiva il bottone “Salva”.
  3. L'utente clicca il bottone “Salva”.
    - a. Il sistema mostra un errore e chiede di modificare data e durata.
- L'interazione riprende dal passo 2.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

## 9.4.2. Read

La lettura degli interventi è intesa in questo contesto come operazione di recupero dal database dei dati degli interventi relativi ad una determinata sessione. I dati saranno quindi visualizzati all'utente, esattamente come è stato fatto per gli eventi e le sessioni.

### Server Node

Al server, nel file contenente le API esposte, è stata aggiunta quella per l'estrazione dal database di tutti e soli gli eventi inerenti ad una sessione, quella il cui riferimento è stato inviato come parametro nel body della richiesta html dal server Ionic.

L'API (**/intervents**) richiede che il body della richiesta sia completa di riferimento all'oggetto Sessione di cui si vogliono recuperare gli interventi. Effettua una richiesta al database che estrae la lista degli interventi che soddisfano i vincoli per rimandarla al mittente.

La lettura degli interventi è stata testata chiamando l'API che estrae gli interventi per una delle sessioni precedentemente create tramite Postman.

- Test di Lettura degli Interventi

The screenshot shows a Postman interface with the following details:

- Request URL:** http://192.168.0.44:8080/api/intervents
- Method:** POST
- Body:** x-www-form-urlencoded
- Key:** session  
Value: 57b423edd0aa8c051a8e86d3

**Response:**

- Status: 200 OK
- Time: 27 ms
- JSON Response:**

```
1  [
2   "success": true,
3   "data": [
4     {
5       "id": "57b42460d0aa8c051a8e86d5",
6       "title": "Primo intervento di test",
7       "date": "2016-10-15T16:00:00.000Z",
8       "duration": 60,
9       "speaker": "speaker1",
10      "session": "57b423edd0aa8c051a8e86d3",
11      "status": "programmed",
12      "_v": 0
13    },
14    {
15      "id": "57b42470d0aa8c051a8e86d7",
16      "title": "Terzo intervento di test",
17      "date": "2016-10-15T16:00:00.000Z",
18      "duration": 60,
19      "speaker": "speaker3",
20      "session": "57b423edd0aa8c051a8e86d3",
21      "status": "programmed",
22      "_v": 0
23    },
24    {
25      "id": "57b42481d0aa8c051a8e86d0",
26      "title": "Secondo intervento di test",
27      "date": "2016-10-15T16:00:00.000Z",
28      "duration": 60,
29      "speaker": "speaker2",
30      "session": "57b423edd0aa8c051a8e86d3",
31      "status": "programmed",
32      "_v": 0
33    }
34  ]
35 ]
```

## Server Ionic

Dal punto di vista dell'utente, gli interventi saranno presentati come lista con le informazioni fondamentali nelle pagine in cui è visualizzata la lista degli eventi, in particolare nel sottomenù ad albero inserito in ogni card rappresentante un evento. Inoltre è stato aggiunto il servizio per la comunicazione con il server Node alla collezione relativa alla gestione degli eventi.

home-page.html - personal-home-page.html - personal-events-page.html

Queste pagine html sono state completate con l'aggiunta della lista degli interventi per ciascuna sessione nel menù ad albero che l'utente espande per visualizzare rapidamente più dettagli dell'evento.

```
<ion-list *ngIf="event.expanded">
  <ion-item text-wrap *ngFor="let session of event.sessions">
    <p>
      <ion-icon *ngIf="!session.expanded" name="arrow-dropdown"
(click)="session.expanded=!session.expanded"></ion-icon>
      <ion-icon *ngIf="session.expanded" name="arrow-dropup"
(click)="session.expanded=!session.expanded"></ion-icon>
      {{session.title}} - {{session.startDate}}
    </p>
    <ion-list *ngIf="session.expanded">
      <ion-item text-wrap *ngFor="let intervent of session.intervents">
        <p>
          {{intervent.title}} - {{intervent.date}} --- {{intervent.status}}
        </p>
      </ion-item>
    </ion-list>
  </ion-item>
</ion-list>
```

home-page.ts - personal-home-page.ts - personal-events-page.ts

Gli script controllano le pagine modificate poco prima, includono nei metodi per il recupero delle informazioni su un evento, le istruzioni per inserire oltre alla lista delle sessioni per ogni intervento, la lista degli interventi per ogni sessione. Il metodo finito risulta quindi il seguente.

```
updateEvents(){
  let _events = [];
  this.es.getPublicEvents().map(res=> res.json()).subscribe((data) => {
    data.data.forEach(event =>{
      event.expanded=false;
      let _sessions = [];
```

```

this.es.getSessions(event._id).map(res=>res.json()).subscribe(data=>{
  data.data.forEach(session =>{
    session.expanded=false;
    //Cerca e unisce gli INTERVENTI
    let _intervents = [];
    this.es.getIntervents(session._id).map(res=>res.json()).subscribe(data=>{
      data.data.forEach(intervent =>{
        _intervents.push(intervent);
        session.intervents=_intervents;
      })
    });
    _sessions.push(session);
    event.sessions=_sessions;
  })
});
_events.push(event);
this.events = _events;
});
});
}

```

session-page/session-page.html - session-page.ts

La pagina che contiene i dettagli di una singola sessione è stata completata con la lista degli interventi di quella sessione poichè la caratterizzano nel modello logico previsto.

```

<ion-list>
  <ion-toolbar padding>
    <ion-title>
      INTERVENTI
    </ion-title>
    <ion-buttons end>
      <button (click)="newIntervent()">
        <ion-icon name="add"> Nuovo Intervento</ion-icon>
      </button>
    </ion-buttons>
  </ion-toolbar>
  <ion-card *ngFor="let intervent of interventions">
    <ion-card-header>
      <ion-toolbar>
        <ion-title>
          {{intervent.title}}
        </ion-title>
        <ion-buttons end>
          <!-- OPERAZIONI SUGLI INTERVENTI -->
        </ion-buttons>
      </ion-toolbar>
      Data: {{intervent.date}} - ({{{intervent.duration}}} minuti)
    </ion-card-header>
  </ion-card>
</ion-list>

```

Lo script che controlla la pagina sopra è stato integrato con l'operazione di recupero degli interventi della sessione in questione come è stato fatto nella pagina di un evento per il recupero della lista delle sessioni.

services/event-services.ts

Il servizio aggiunto costruisce come al solito la richiesta html e restituisce il risultato, ottenuto dal server Node, come Observable al chiamante.

Test

La lettura degli interventi è stata testata verificando che il sistema visualizzasse correttamente tutto quello che veniva richiesto dall'utente.

<b>Caso d'uso:</b> Visualizzare Interventi	
<b>Descrizione</b>	L'utente visualizza la lista degli interventi di una sessione di un evento pubblico disponibile.
<b>Pre-condizioni</b>	NESSUNA
<b>Post-condizioni per Successo</b>	L'utente visualizza nella pagina la lista degli interventi della sessione scelta per l'evento pubblico scelto.
<b>Post-condizioni per Fallimento</b>	L'utente non visualizza nessuna lista nella pagina.
<b>Evento innescante</b>	L'utente espande il menù ad albero per una sessione nella lista.
<b>Attori primari</b>	Qualsiasi utente.

#### **Scenario o (default):**

1. L'utente clicca sulla freccetta verso il basso del menù ad albero di una sessione.
  - a. Il sistema espande l'albero mostrando la lista di tutti gli interventi della sessione.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

#### 9.4.3. Update

L'aggiornamento delle informazioni di un evento è una funzione dovuta all'utente dal momento che in una situazione reale l'organizzazione dell'evento nel suo complesso potrebbe cambiare in ogni sua parte. In questo caso si prevede che l'operazione sia accessibile

dall'organizzatore dell'evento e dal relatore che tiene l'intervento. Per il momento però, è stata implementata solo per l'organizzatore.

## Server Node

Sul server Node è stata aggiunta l'API (**/updateintervent**) per la modifica delle informazioni nel documento che rappresenta uno specifico evento. Dopo la verifica della presenza di tutte le informazioni necessarie, il server imposta nel documento i nuovi valori dei dati, aggiorna poi la lista degli speakers della sessione a cui afferisce l'intervento e salva il documento aggiornato.

È stato testato anche l'aggiornamento di un intervento utilizzando come oggetto di test uno degli interventi creati tramite Postman, la verifica della corretta scrittura dei dati è stata effettuata tramite Robomongo. Se la modifica include anche il relatore, la lista di relatori nella Sessione afferente deve essere correttamente aggiornata, anche questo è stato verificato grazie al tool.

- Test di Modifica di un Intervento

The screenshot shows the Postman application interface. A POST request is being made to `http://192.168.0.44:8080/api/updateintervent`. The 'Body' tab is selected, showing the following JSON payload:

```
POST http://192.168.0.44:8080/api/updateintervent
{
  "title": "Secondo Intervento MODIFICATO",
  "date": "2016-10-15T16:00",
  "duration": 45,
  "speaker": "SpeakerCambiato",
  "id": "57b43d195d4c08bc3245576a"
}
```

The 'Body' tab also includes a 'key' field which is currently empty. The 'Headers' tab shows 10 headers. The 'Tests' tab is empty. The status bar at the bottom indicates a successful response: `Status: 200 OK Time: 23 ms`.

The screenshot shows the Robomongo interface with the 'LocalConnection' expanded to show 'scribaDB' and its 'Collections' (events, intervenents, sessions, users). The 'intervents' collection is selected in the main pane, showing a list of documents. One document is expanded, revealing fields like '\_id', 'title' (set to 'Secondo Intervento MODIFICATO'), 'date', 'duration', 'speaker', 'session', 'status', and '\_v'. The document ID is ObjectId("57b43d195d4c08bc3245576a").

- Verifica di Aggiornamento della Sessione (speakers)

The screenshot shows the Robomongo interface with the 'LocalConnection' expanded to show 'scribaDB' and its 'Collections' (events, intervenents, sessions, users). The 'sessions' collection is selected in the main pane, showing a list of documents. One document is expanded, revealing fields like '\_id', 'title' (set to 'sessione di test 2'), 'startDate', 'endDate', 'status', 'event', and 'speakers' (an array containing four elements: [0] [pepponefx], [1] speaker1, [2] SpeakerCambiato, [3] speaker3). The document ID is ObjectId("57b43cf25d4c08bc32455768").

## Server Ionic

All'utente, l'operazione di modifica dei dati di un intervento è accessibile, come per gli eventi e per le sessioni, dalla pagina dedicata all'oggetto. In questo caso si tratta della pagina dedicata al singolo evento che espone le operazioni per la sua modifica. Per accedere alla suddetta pagina, alla pagina dedicata alla sessione di cui fa parte l'intervento, sono stati aggiunti i comandi per l'accesso ai dettagli del singolo intervento.

session-page/session-page.html - session-page.ts

La pagina mostra la lista degli interventi di cui si compone la sessione. A ciascuno di essi è stato aggiunto il bottone per aprire la pagina dell'intervento per visualizzarne e modificarne i dettagli.

Lo script associato alla pagina è stato arricchito con l'implementazione del metodo che apre la pagina di dettaglio di un intervento scelto dall'utente come è stato fatto per i singoli eventi e sessioni.

intervent-page/intervent-page.html - intervent-page.ts

La pagina mostra i dettagli dell'intervento e il campo che durante il corso dello stesso si riempirà del testo trascritto corrispondente al parlato del relatore. Come per le sessioni e gli eventi, contiene un form nascosto che, sotto attivazione dell'utente mostrerà i campi modificabili.

Lo script si occupa di gestire il comportamento della pagina, in sprint successivi si interfacerà con il servizio di riconoscimento del parlato. Per il momento contiene solo i controlli per la modifica dei dati dell'intervento come quello che verifica se l'evento è concomitante con un altro (overlap).

services/event-services.ts

Il servizio aggiunto, anche in questo caso, si occupa di costruire la richiesta di modifica da inviare al server Node. I risultati di tale richiesta saranno comunicati al chiamante tramite un Observable.

## Test

L'aggiornamento di un intervento è stato testato sul lato client simulando dei possibili scenari di interazione di un utente con il sistema controllando che le risposte di quest'ultimo fossero adeguate alla circostanza.

Caso d'uso: Modificare Intervento	
<b>Descrizione</b>	L'utente modifica i dati di un intervento di una sessione di un evento pubblico da lui creato.
<b>Pre-condizioni</b>	L'utente deve aver aperto la pagina dell'intervento da modificare; l'intervento deve essere in stato “programmed”.
<b>Post-condizioni per Successo</b>	L'intervento viene modificato e salvato nel database con le nuove informazioni.

<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone “Modifica”.
<b>Attori primari</b>	Organizzatore dell'evento.

#### **Scenario 0 (default):**

1. L'utente clicca sul bottone “Modifica”.
  - a. Il sistema mostra il form per la modifica dei dati.
2. L'utente modifica uno o più campi e clicca il bottone “Salva”.
  - a. Il sistema modifica le informazioni e ricarica la pagina precedente.

#### **Scenario 1**

2. L'utente modifica uno o più campi in maniera errata
  - a. Il sistema DISATTIVA il bottone “Salva”

#### **Scenario 2**

2. L'utente modifica le date in maniera tale che l'intervento risulti concomitante con un altro e clicca su “Salva”.
  - a. Il sistema mostra un errore e chiede di modificare i dati.
  - L'interazione riprende dal passo 2.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

#### 9.4.4. Delete

Anche l'eliminazione di un intervento è stata prevista come operazione accessibile sia dall'organizzatore dell'evento, sia dal relatore dello specifico intervento che in una situazione reale potrebbe venir meno e non partecipare all'evento. Per il momento però, la funzione è stata implementata solo per l'organizzatore.

##### Server Node

Sul lato server, ora che gli oggetti che costituiscono un evento sono completamente implementati, oltre all'inserimento dell'API di eliminazione di un intervento (**/deleteintervent**), sono state completate le API di eliminazione di una sessione e di un evento nel medesimo file che le ospita tutte.

L'eliminazione di un intervento viene effettuata cercando e cancellando il documento identificato con la stringa passata come parametro. Vengono poi aggiornati i relatori in lista

della sessione a cui afferisce l'intervento. L'eliminazione di una sessione è stata aggiornata per eliminare anche tutti gli interventi ad essa afferenti. Allo stesso modo l'eliminazione di un evento avviene cancellando prima tutte le sessioni ed interventi ad esso associati.

L'eliminazione di un intervento comporta un impatto sulla lista dei relatori coinvolti in una sessione. Il funzionamento dell'API è stato testato tramite Postman mentre la verifica dei dati, al solito è stata effettuata tramite il tool Robomongo. Naturalmente anche l'eliminazione di una singola sessione e dell'intero evento sono state testate nuovamente per verificarne il corretto funzionamento dopo le modifiche apportate.

- Test di Eliminazione di un Intervento

The screenshot shows a POST request in Postman to the URL `http://192.168.0.44:8080/api/deleteintervent`. The request body is set to `x-www-form-urlencoded` and contains a single key-value pair: `id` with value `57b43d195d4c08bc3245576a`. The response tab shows a successful `200 OK` status with a response body of `{"success": true, "msg": "Eliminazione Intervento eseguita"}`.

L'intervento eliminato non risulta più presente tra i documenti della collection del database.

- Verifica di Aggiornamento della Sessione (speakers)

Key	Type
1	Object
2	Object
3	Object
4	Object
5	Object
6	Object
7	Object
8	Object
9	Object
10	Object
11	Object
12	Object
13	Object
_id	Object
title	String
startDate	Date
endDate	Date
status	String
event	String
speakers	Array
[0]	String
[1]	String
[2]	String
_v	Int32

Come è possibile notare, tra gli speakers non è più presente l'username di chi avrebbe tenuto l'intervento eliminato.

- Test di Eliminazione di una Sessione

```

1 [
2   "success": true,
3   "msg": "Eliminazione Sessione e relativi Interventi eseguita"
4 ]
    
```

Nella collection delle sessioni manca effettivamente il documento eliminato, così come in quella degli interventi mancano gli interventi creati per quella sessione.

- Test di Eliminazione di un Evento

The screenshot shows the Postman interface. At the top, the URL is set to `http://192.168.0.44:8080/api`. The method is selected as `POST`, and the endpoint is `http://192.168.0.44:8080/api/deleteevent`. The `Body` tab is active, showing a single parameter named `id` with the value `57b43ce65d4c08bc32455766`. Below the body, the `Body` tab is selected again, showing the JSON response: 

```
1 [ { "success": true, 2   "msg": "Eliminazione Evento e relative Sessioni e Interventi eseguita" 3 } ] 4 }
```

. The status bar at the bottom indicates `Status: 200 OK` and `Time: 44 ms`.

Anche in questo caso nel database MongoDB risulta essere stato cancellato correttamente il documento relativo all'evento scelto e tutte le sessioni ad esso afferenti così come gli interventi di tali sessioni.

## Server Ionic

Sul lato utente sono state esposte anche per l'eliminazione le funzioni necessarie, in particolare è stato aggiunto l'apposito bottone nella pagina contenente la lista degli interventi e nello script la funzione che chiama il servizio che è stato inserito nello script apposito.

session-page/session-page.html - session-page.ts

Tra i bottoni disponibili per ogni card rappresentante ciascuna un intervento diverso è stato aggiunto il bottone per l'eliminazione tra le operazioni possibili.

Lo script che gestisce i componenti dinamici della pagina è stato arricchito con la funzione di eliminazione di un intervento che, dopo aver chiesto conferma all'utente, chiama il servizio che comunica al server Node di cancellare l'intervento.

services/event-services.ts

Il servizio aggiunto come segue, prepara la solita richiesta html inviandola successivamente al server Node. Ricevuta la risposta, la incorpora in un oggetto Observable e la restituisce al chiamante.

## Test

Come per tutte le altre operazioni anche l'eliminazione di un intervento è stata testata simulando l'interazione dell'utente con il sistema.

<b>Caso d'uso:</b> Cancellare Intervento	
<b>Descrizione</b>	L'utente elimina un intervento di una sessione di un evento da lui creato.
<b>Pre-condizioni</b>	L'utente deve aver aperto la pagina della sessione per cui desidera eliminare un intervento.
<b>Post-condizioni per Successo</b>	L'utente visualizza la lista degli interventi aggiornata (senza l'intervento eliminato).
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone “Elimina”.
<b>Attori primari</b>	Organizzatore dell'evento.

#### **Scenario o (default):**

1. L'utente clicca sul bottone “Elimina” di un intervento.
  - a. Il sistema chiede conferma.
2. L'utente conferma.
  - a. Il sistema elimina l'intervento e aggiorna la pagina.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

# 10. Sprint 5 - Completamento del Core-Goal

Durante il meeting alla fine dello sprint precedente, sono stati eseguiti i test alla presenza dell'intero team che pur accettando il funzionamento delle operazioni implementate, ha riscontrato qualche imperfezione dal punto di vista dell'aspetto grafico dell'interfaccia. Per questo motivo sono stati create ulteriori issue etichettate come “enhancement” e collocate nella coda “product backlog” e quindi programmati per sprint futuri.

Per quanto riguarda invece le issue completate, sono state spostate nella coda “done” mentre la coda “sprint backlog”, contenente le issue da implementare entro lo sprint successivo, è stata riempita con nuove issue più urgenti che vanno a completare il goal principale dell'applicazione. In particolare sono stati programmati l'avvio e l'arresto di un intervento da parte del relatore od organizzatore dell'evento; l'ascolto di un intervento da parte del partecipante ad un evento e la rilettura di un evento concluso.

In questo capitolo saranno illustrate le tecniche utilizzate per completare questi task, sarà analizzato il codice prodotto e saranno illustrati i test effettuati per ciascuno dei task.

## 10.1. Osservazione Eventi

Per “osservazione eventi” si intende la manifestazione di interesse da parte di un utente per un evento, in particolare l'utente può tenere sotto controllo l'evento poiché interessato ad esso e ritrovarlo facilmente nel momento in cui vorrà ascoltarlo. Ovviamente un utente potrebbe nel tempo cambiare idea e perdere l'interesse per l'evento quindi decidere di non tenerlo più in osservazione.

Per l'analisi dello sviluppo di questo task si utilizzerà il medesimo schema visto in precedenza, ovvero sarà illustrato il cambiamento apportato rispettivamente al database, al server Node (con relativi test) e al server Ionic (con relativi test).

### Database

Sul database la modifica è stata semplice grazie alla possibilità che offre MongoDB di aggiungere campi ai documenti senza intaccare l'integrità dei dati. In questo caso è stato aggiunto un campo alla struttura del documento “Utente” che tiene traccia degli eventi osservati.

- **observedEvent:** array di stringhe che contiene i riferimenti (ObjectID) agli eventi che l'utente vuole tenere in osservazione.

## Server Node

Sul lato server è stato innanzitutto aggiornato lo schema dell'oggetto Utente in modo da essere conforme all'aggiunta effettuata sul database aggiungendo un attributo allo schema. In secondo luogo sono state create le API per la scrittura di un evento osservato e per l'eliminazione di un evento tra quelli osservati. Inoltre è stata predisposta l'API di lettura della lista di tutti gli eventi osservati da un determinato utente.

user.js

```
[...attributi dello schema...]
observedEvents: {
    type: Array,
    required: false
}
[...eventuali altri attributi...]
```

server.js

Le API necessarie al server per completare il task sono state aggiunte al file che controlla l'indirizzamento alle funzioni.

La prima (**/addobservedevent**) rappresenta la funzione di aggiunta al documento dell'utente di cui viene richiesto il riferimento, del codice identificativo dell'evento che intende osservare dopo aver controllato che non sia già presente.

L'API **/removeobservedevent** invece non fa altro che recuperare il documento relativo all'utente indicato e rimuovere dalla lista di eventi osservati, il riferimento all'evento che non interessa più.

In ultima istanza, dato il riferimento ad un utente, l'API **/observedevents** restituisce l'array di eventi che l'utente tiene sotto osservazione.

Le API sono state testate creando degli eventi di prova e aggiungendone alcuni alla lista di quelli osservati per poi provare a recuperare la lista così ottenuta. Tutte le chiamate saranno simulate tramite il tool Postman e nel database saranno mostrate le liste aggiornate grazie all'utilizzo di Robomongo per l'ispezione dei dati.

- Test di Aggiunta di un Evento a quelli Osservati

The top portion of the image shows a POST request in Postman to `http://192.168.0.44:8080/api/addobservedevent`. The request body contains a single key-value pair: `id: 57b591ca94b4617e27f11b45`. The response status is 200 OK, and the JSON response is:

```

1  {
2     "success": true,
3     "data": {
4         "_id": "57b07806c4e991283102119f",
5         "name": "Giuseppe_updated",
6         "surname": "Iaffaldano_updated",
7         "username": "pepponefx",
8         "password": "$2a$10$181p41T6E4skx4xbRRp8MORpMbm4iUn8IAirFhcB68yEfn3xDkVC",
9         "email": "email.updated@gmail.com",
10        "v": 2,
11        "joinedEvents": [],
12        "observedEvents": [
13            "57b591c694b4617e27f11b43",
14            "57b591c894b4617e27f11b44",
15            "57b591ca94b4617e27f11b45"
16        ]
17    }
18 }

```

The bottom portion of the image shows Robomongo 0.9.0-RC9. On the left, the database structure is visible with collections: LocalConnection (2), System, scribaDB (Collections: events, intervents, sessions, users), Functions, and Users. In the center, a query is run against the `users` collection: `db.getCollection('users').find({})`. The results show the updated user document with the new observedEvents array.

Key	Value	Type
<code>_id</code>	<code>ObjectId("57b07806c4e991283102119f")</code>	<code>ObjectID</code>
<code>name</code>	<code>Giuseppe_updated</code>	<code>String</code>
<code>surname</code>	<code>Iaffaldano_updated</code>	<code>String</code>
<code>username</code>	<code>pepponefx</code>	<code>String</code>
<code>password</code>	<code>\$2a\$10\$181p41T6E4skx4xbRRp8MORpMbm4iUn8IAirFhcB68yEfn3xDkVC</code>	<code>String</code>
<code>email</code>	<code>email.updated@gmail.com</code>	<code>String</code>
<code>joinedEvents</code>	<code>[0 elements]</code>	<code>Array</code>
<code>observedEvents</code>	<code>[3 elements]</code>	<code>Array</code>
<code>  [0]</code>	<code>57b591c694b4617e27f11b43</code>	<code>String</code>
<code>  [1]</code>	<code>57b591c894b4617e27f11b44</code>	<code>String</code>
<code>  [2]</code>	<code>57b591ca94b4617e27f11b45</code>	<code>String</code>
<code>v</code>	<code>3</code>	<code>Int32</code>

- Test di Eliminazione di un Evento da quelli Osservati

The screenshot shows the Postman interface with the following details:

- URL:** http://192.168.0.44:8080/api/removeobservedevent
- Method:** POST
- Body (x-www-form-urlencoded):**
  - id: 57b591c894b4617e27f11b44
- Response Status:** 200 OK, Time: 26 ms
- Response Body (Pretty JSON):**

```

1  {
2   "success": true,
3   "data": {
4     "_id": "57b07806c4e991283102119f",
5     "name": "Giuseppe_updated",
6     "surname": "Iaffaldano_updated",
7     "username": "pepponefx",
8     "password": "$2a$10$181p41T6E4skx4xbRRp8MORpKbm4iUnBIAirFhcB68yEfn3xDKvC",
9     "email": "email.updated@gmail.com",
10    "v": 3
11   "joinedEvents": [],
12   "observedEvents": [
13     "57b591c694b4617e27f11b43",
14     "57b591ca94b4617e27f11b45"
15   ]
16 }
17 
```

The screenshot shows the Robomongo interface with the following details:

- Database Structure:**
  - LocalConnection (2)
    - System
    - scribaDB
      - Collections (4)
        - events
        - intervents
        - sessions
        - users
- Query Result Table:**

Key	Value	Type
_id	ObjectId("57988c17989bb54...")	Object
name	Giuseppe_updated	String
surname	Iaffaldano_updated	String
username	pepponefx	String
password	\$2a\$10\$181p41T6E4skx4xbRRp8MORpKbm4iUnBIAirFhcB68yEfn3xDKvC	String
email	email.updated@gmail.com	String
joinedEvents	[0 elements]	Array
observedEvents	[2 elements] <ul style="list-style-type: none"> <li>[0] 57b591c694b4617e27f11b43</li> <li>[1] 57b591ca94b4617e27f11b45</li> </ul>	Array
v	4	Int32

- Test di Lettura della lista di Eventi Osservati

```

1 [ { "success": true,
2   "data": [
3     {
4       "id": "57b591c694b4617e27f11b43",
5       "title": "evento di test",
6       "startDate": "2016-10-15T15:30:00.000Z",
7       "endDate": "2016-10-15T18:30:00.000Z",
8       "location": "Uniba - Via Orabona, 4",
9       "status": "programmed",
10      "organizer": "pepponefx",
11      "public": true,
12      "v": 0,
13      "allowed": []
14    },
15    {
16      "id": "57b591ca94b4617e27f11b45",
17      "title": "evento di test 3",
18      "startDate": "2016-10-15T15:30:00.000Z",
19      "endDate": "2016-10-15T18:30:00.000Z",
20      "location": "Uniba - Via Orabona, 4",
21      "status": "programmed",
22      "organizer": "pepponefx",
23      "public": true,
24      "v": 0,
25      "allowed": []
26    }
27  ]
28 }
29 ]
  
```

## Server Ionic

Sul lato client (server Ionic) le funzioni sono state esposte all'utente perché possa usufruirne. Come sarà mostrato di seguito, sono stati aggiunti dei buttoni iconici per permettere ad un utente autenticato di decidere di seguire un evento che visualizza nella lista di eventi disponibili o di smettere di seguirne uno. Inoltre è stata aggiunta una pagina appositamente per la visualizzazione della lista dei soli eventi seguiti in modo che l'utente possa ritrovarli facilmente o rimuoverne. Dal punto di vista di controllo invece, sono state aggiunti i servizi per la comunicazione con il server Node, in particolare quello per aggiungere un evento a quelli osservati, quello per rimuoverne uno e quello per estrarre la lista di eventi. Il modello locale dell'utente è stato infine arricchito con la lista degli eventi osservati per mantenere coerenza nelle informazioni e maggior controllo.

services/models/user-model.ts

Al modello dell'utente è stato aggiunto l'attributo “observedEvents” opzionale:

```

[...]
public observedEvents? :string,
[...]
  
```

services/user-services.ts

Al file che gestisce le comunicazioni con il server che afferiscono ad un utente, sono stati aggiunti i servizi di creazione e invio delle richieste per l'aggiunta o rimozione di un evento tra quelli osservati che passano il riferimento all'evento stesso.

services/event-services.ts

Al file che gestisce invece le comunicazioni con il server che coinvolgono gli oggetti evento, è stato aggiunto il servizio di lettura della lista degli eventi osservati dall'utente autenticato.

personal-home-page/personal-home-page.html - personal-home-page.ts

Alla pagina è stato aggiunto il bottone per aggiungere/rimuovere dalla lista di eventi osservati un determinato elemento tra gli eventi disponibili. Il bottone cambia aspetto grazie alla funzione IF offerta dal framework Angular e grazie alle operazioni delegate allo script di controllo della pagina.

Lo script associato implementa, in aggiunta alle altre, la funzione di chiamata al servizio di aggiunta/rimozione di un evento tra quelli osservati dall'utente autenticato.

observed-events-page/observed-events-page.html - observed-events-page.ts

La pagina è strutturata come le altre per mantenere coerenza nell'interfaccia. Contiene la lista degli eventi osservati da un utente con le informazioni fondamentali e il bottone per rimuoverli dalla lista.

Lo script controlla le operazioni di rimozione dalla lista di eventi osservati e di aggiornamento della pagina chiamando i servizi appositi precedentemente illustrati.

## Test

Come per lo sprint precedente, sono stati svolti anche i test preliminari del sistema simulando l'interazione dell'utente durante l'uso del sistema secondo lo schema seguente.

Caso d'uso: Osservare Evento	
<b>Descrizione</b>	L'utente aggiunge un evento alla lista di eventi osservati.
<b>Pre-condizioni</b>	L'utente deve aver aperto la pagina degli eventi disponibili ("Tutti gli Eventi") dal menù laterale.
<b>Post-condizioni per Successo</b>	L'evento è stato aggiunto alla lista di quelli osservati e l'utente visualizza l'icona per smettere di osservarlo.
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.

<b>Evento innescante</b>	L'utente clicca il bottone "Osserva".
<b>Attori primari</b>	qualsiasi utente.

#### Scenario o (default):

1. L'utente clicca sul bottone "Osserva" di un evento.
  - a. Il sistema aggiunge l'evento alla lista di quelli osservati e modifica l'icona.

<b>Caso d'uso:</b> Smettere di Osservare Evento	
<b>Descrizione</b>	L'utente rimuove un evento dalla lista di eventi osservati.
<b>Pre-condizioni</b>	L'utente deve aver aperto la pagina degli eventi disponibili ("Tutti gli Eventi") oppure la pagina degli eventi osservati ("Eventi Osservati") dal menù laterale.
<b>Post-condizioni per Successo</b>	L'evento è stato rimosso dalla lista di quelli osservati e l'utente visualizza l'icona per osservarlo.
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone "Non osservare più".
<b>Attori primari</b>	qualsiasi utente.

#### Scenario o (default):

1. L'utente clicca sul bottone "Non osservare più" di un evento.
  - a. Il sistema rimuove l'evento dalla lista di quelli osservati e modifica l'icona.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

## 10.2. Avvio Eventi

Quello che viene illustrato in questa sezione rappresenta il fulcro del sistema, il punto di contatto effettivo tra un relatore e i partecipanti, il momento in cui chi parla comunica con chi ascolta attraverso il sistema stesso. Il task si compone di diverse fasi, innanzitutto chi si trova ad interagire dal lato del relatore (necessariamente da ambiente Desktop), seleziona un intervento da avviare, a questo punto dovrebbe permettere ai partecipanti di accedere

all'evento, in particolar modo all'intervento, per cui “apre la stanza”. Da questo momento i partecipanti potranno accedere alla stanza virtuale in attesa che inizi la trasmissione dei dati, ossia in attesa che il relatore inizi a parlare. Il relatore deve quindi abilitare il servizio di trascrizione e iniziare il suo discorso. Al termine del discorso, il relatore deve scegliere se tenere in memoria il testo trascritto e quindi chiudere la stanza oppure eliminare tutti i dati e tornare ad uno stato precedente.

In sintesi, il client Web (tramite il server Ionic) chiederà al server Node di aprire la stanza virtuale, il server aprirà la socket in ascolto su una porta libera che andrà a memorizzare nel database. I client mobile saranno quindi in grado di collegarsi alla stanza virtuale e ricevere i messaggi di trascrizione).

Per questo task non sono state apportate modifiche al database, la struttura seguita è la seguente: verranno illustrate le modifiche sul server Node e poi sul lato client (server Ionic). Ad ogni passo seguiranno i relativi test.

## Server Node

Sul server Node è stata implementata allo scopo una sola API che si occupa di gestire l'interazione durante tutta la “vita” di un intervento. Essa si occupa di aprire un nuovo server socket.io su una porta libera perché la comunicazione durante l'intero intervento possa avvenire su quella porta. Quello che è stato realizzato durante gli sprint preliminari è stato quindi inglobato in questa API per gestire messaggi in entrata e in uscita oltre che le connessioni dei diversi partecipanti.

### server.js

Alla chiamata della prima API, che richiede l'identificativo dell'intervento che si sta avviando, si occupa di aprire una socket in ascolto sulla porta “0” ovvero si lascia al sistema il compito di individuare una porta TCP libera su cui avviare la comunicazione. Individuata la porta, essa viene inserita nell'apposito campo del documento identificante l'intervento nel database e viene modificato lo stato dello stesso in “ongoing” (in corso). A questo punto chiunque saprà che è possibile accedere all'evento poiché risulterà pronto per iniziare.

La socket aperta implementa i listener per la connessione di un client, per la ricezione di messaggi dallo speaker (che vengono inoltrati a tutti i client connessi), per la disconnessione di un singolo partecipante e per la chiusura della stanza che disconnette tutti i partecipanti dandone previo avviso e chiude la connessione. La seconda API invece, salva semplicemente il testo inviato nel body come parametro, nel documento dell'intervento nel database.

```

apiRoutes.post('/openerver', function(req, res) {
  var token = getToken(req.headers);
  if (token) {
    //DEPENDENCIES
    var server = require("http").Server(express);
    var io = require("socket.io")(server);
    //TEST PORTS
    server.listen(0);
    //SAVE PORT IN INTERVENT
    Intervent.findOne({ _id: req.body.id }, function (err, intervent){
      if(err) throw err;

      intervent.status= 'ongoing';
      intervent.port= server.address().port;
      intervent.save();
      Session.findOne({ _id: intervent.session}, function (err, session){
        if(err) throw err;

        session.status= 'ongoing';
        session.save();
        Event.findOne({ _id: session.event }, function (err, event){
          if(err) throw err;

          event.status= 'ongoing';
          event.save();
        });
      });
    });
  });

  //LISTENER
  var allClients=[];
  var _transcription="";
  io.on('connection', function (socket) {
    allClients.push(socket);

    socket.on('client_type',function(data){
      var msg="User Type: ";
      msg += data.text;
      if(data.text=='Listener'){
        socket.emit('previous_text', {text: _transcription})
      }
    });

    socket.on('client_message',function(data){
      _transcription += data.text;
      socket.broadcast.emit('server_message',{text:data.text});
    });

    socket.on('close_room', function (data) {
      socket.broadcast.emit('closing_room', {text:"La stanza è stata
Chiusa!"});
      allClients.forEach(function(s) {
        s.disconnect();
      });
      //RIAGGIORNA LO STATO DI INTERVENTO; SESSIONE ED EVENTO
    });
  });
}

```

```

        Intervent.findOne({ _id: req.body.id }, function (err, intervent){
            if(err) throw err;

            intervent.status= 'programmed';
            intervent.save();
            Session.findOne({ _id: intervent.session}, function (err, session){
                if(err) throw err;

                session.status= 'programmed';
                session.save();
                Event.findOne({ _id: session.event }, function (err, event){
                    if(err) throw err;

                    event.status= 'programmed';
                    event.save();
                });
            });
            io.close();
        });
    });

    res.send({success: true, port: server.address().port});
} else {
    return res.status(403).send({success: false, msg: 'Nessun token ricevuto'});
}
});

apiRoutes.post('/saveinterventtext', function(req, res) {
    var token = getToken(req.headers);
    if (token) {
        Intervent.findOne({
            _id: req.body.id
        }, function(err, intervent) {
            if (err) throw err;

            if (!intervent) {
                return res.send({success: false, msg: 'Intervento non trovato'});
            } else {
                intervent.text=req.body.text;
                intervent.save();

                res.json({success: true, data: intervent});
            }
        });
    } else {
        return res.status(403).send({success: false, msg: 'Nessun token ricevuto'});
    }
});

```

Anche questa API è stata testata per quanto riguarda l'attivazione del server socket e la scrittura dello stato e della porta nel database, per quanto riguarda invece il test del server

socket vero e proprio, è stato testato durante la fase successiva di implementazione del lato Ionic.

- Test di Apertura della Stanza

The screenshot displays two applications used for testing and monitoring a MongoDB database.

**Postman:** A tool for making HTTP requests. The request is a POST to `http://192.168.0.44:8080/api/openerver`. The body contains a single form-data field named `id` with value `57b6b80f934bf5c81c518a02`. The response status is `200 OK` with a time of `355 ms`. The JSON response is:

```

1  {
2   "success": true,
3   "port": 41490
4 }

```

**Robomongo:** A MongoDB interface. It shows a local connection to `localhost:27017` named `scribaDB`. A query is run against the `intervents` collection: `db.getCollection('intervents').find({})`. The results show one document with the following fields and values:

Key	Type	Value
<code>_id</code>	Object	<code>ObjectId("57b6b80f934bf5c81c518a02")</code>
<code>title</code>	String	Intervento di test
<code>date</code>	Date	2016-10-15 16:00:00.000Z
<code>duration</code>	Int32	60
<code>speaker</code>	String	speaker1
<code>session</code>	String	57b6b7fe934bf5c81c518a01
<code>status</code>	String	ongoing
<code>v</code>	Int32	0
<code>port</code>	Int32	41490

- Test di Salvataggio del Testo

The screenshot shows two applications demonstrating the API's functionality.

**Postman API Test Results:**

- Method: POST
- URL: http://192.168.0.44:8080/api/saveinterventext
- Body tab selected, showing form-data fields:
  - id: 57b6b80f934bf5c81c518a02
  - text: Testo di esempio da salvare nell'intervento per verificare che l'API funzioni correttamente.
- Response status: 200 OK, Time: 270 ms
- Response body (Pretty JSON):
 

```

1  {
2   "success": true,
3   "data": {
4     "text": "Testo di esempio da salvare nell'intervento per verificare che l'API funzioni correttamente.",
5     "_id": "57b6b80f934bf5c81c518a02",
6     "title": "Intervento di test",
7     "date": "2016-10-15T16:00:00.000Z",
8     "duration": 60,
9     "speaker": "speaker1",
10    "session": "57b6b7fe934bf5c81c518a01",
11    "status": "programmed",
12    "_v": 0,
13    "port": 41490
14  }
15 }
```

**Robomongo MongoDB Interface:**

- File, View, Options, Window, Help menu
- LocalConnection (2) - scribaDB collection
- db.getCollection('intervents').find({}) query
- intervents table results:
 

Key	Type
_id	Object
title	Object
date	Object
duration	Object
speaker	Object
session	Object
status	Object
_v	Object
port	Object
text	String

## Server Ionic

Sul server Ionic sono state apportate semplici modifiche alle pagine e agli script già esistenti, non sono state aggiunte nuove pagine. Nello specifico sarà illustrato come sono stati aggiunti i comandi di avvio di un intervento e come è stato completato lo script che contiene i servizi per la gestione degli eventi a cui sono stati aggiunti nuovi metodi. L'unica aggiunta effettuata è stata un nuovo script per la gestione dei servizi di trascrizione.

intervent-page/intervent-page.html - intervient-page.ts

Alla pagina è stata aggiunta una card che riporterà la trascrizione elaborata e si riempirà man mano che si procede nel parlato. Inoltre sono presenti i comandi per aprire la stanza e per

avviare la trascrizione; sono poi stati aggiunti i comandi per chiudere la stanza e salvare il testo dell'intervento.

Allo script sono stati aggiunti i metodi per gestire le richieste di apertura e chiusura della stanza, le richieste di avvio e stop del riconoscimento comprendenti le istruzioni per iniettare il testo nell'html e i metodi per chiedere il salvataggio del testo dell'intervento.

services/event-services.ts

In questa collezione di servizi sono stati inseriti quelli per richiedere al server Node l'apertura della stanza virtuale e quello per richiedere il salvataggio del testo nel database. La chiusura della stanza infatti avviene tramite comunicazione con la socket.

services/recognition-services.ts

Il servizio di trascrizione non è altro che un porting in TypeScript di quello che è stato realizzato i JavaScript per gli sprint preliminari. In realtà TypeScript è un superset rispetto a JavaScript, questo significa che tutto quello che è scritto in javascript dovrebbe poter essere eseguito da un compilatore TypeScript, tuttavia alcuni elementi vanno organizzati diversamente. Si può notare infatti che i listener degli eventi vengono gestiti in TypeScript come metodi a parte che vengono chiamati al verificarsi di un evento. I listener veri e propri degli eventi sono definiti nel costruttore. Infine, in JavaScript, la tipizzazione non è forte, per cui era facile utilizzare l'oggetto che gestisce il servizio di trascrizione “webkitSpeechRecognition” semplicemente chiamandolo dal “window” ovvero dalla pagina del browser Chrome che lo contiene. Nel caso di TypeScript è stato necessario definire una variabile di tipo arbitrario (equivalente di un generico Object nella programmazione ad oggetti) per rendere coerente il namespace. Per risolvere il problema del livelock di cui si è parlato in sprint precedenti, in fase di porting è stato aggiunto un timer che riavvia la trascrizione ogni qualvolta non si riceve un risultato dal servizio per 3 secondi. Il tempo di attesa sarà successivamente regolato in fase di test per evitare sovraccarichi computazionali.

```
import { Injectable, Inject } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import { Observer } from 'rxjs/Observer';
import { Events } from 'ionic-angular';

declare var webkitSpeechRecognition: any;

@Injectable()
export class TranscriptionService{
  private recognition;
  public final_transcript = "";
  public recognizing = false;
```

```

public delay = null;
public transcriptionChange$: Observable<string>;
private transcriptionObserver: Observer<string>;

constructor(private evts: Events){
    this.transcriptionChange$ = new Observable<string>(observer =>
this.transcriptionObserver = observer).share();
    if ('webkitSpeechRecognition' in window) {
        this.recognition = new webkitSpeechRecognition();
        this.recognition.continuous = true;
        this.recognition.interimResults = true;
        this.recognition.onresult=this.resultHandler.bind(this);
        this.recognition.onstart=this.startHandler.bind(this);
        this.recognition.onerror=this.errorHandler.bind(this);
        this.recognition.onend=this.endHandler.bind(this);
    }else{
        alert("Attenzione, Il servizio è disponibile solo su browser Google Chrome");
    }
}

startDictation() {
    this.final_transcript = "";
    this.recognition.lang = 'it-IT';
    this.recognition.start();
    this.recognizing=true;
}

stopDictation(){
    this.recognition.stop();
    this.recognizing=false;
    this.transcriptionObserver.complete();
}

resetTimeOut(){
    clearTimeout(this.delay);
    this.delay = setTimeout(() => {
        this.recognition.stop();
    }, 3000);
}

resultHandler(event){
    this.resetTimeOut();
    var interim_transcript = '';
    for (var i = event.resultIndex; i < event.results.length; ++i) {
        if (event.results[i].isFinal) {
            this.final_transcript += event.results[i][0].transcript;
            this.transcriptionObserver.next(event.results[i][0].transcript);
        } else {
            interim_transcript += event.results[i][0].transcript;
        }
    };
}

startHandler () {

```

```

        this.recognizing = true;
        this.evts.publish('recognizing', this.recognizing);
    };

    errorHandler(event) {
        console.log("RECOGNITION ERROR: " + event.error);
    };

    endHandler() {
        console.log("RECOGNITION STOPPED");
        if(this.recognizing){
            this.recognition.start();
            console.log("RECOGNITION RESTARTED");
        }
    };
}

```

## Test

Soprattutto in questo caso è stato necessario effettuare un test di simulazione del comportamento dell'utente con il sistema per verificare che le risposte ottenute fossero conformi alle aspettative. Lo schema seguito è quello illustrato sotto.

<b>Caso d'uso:</b> Avviare Intervento	
<b>Descrizione</b>	L'utente avvia un intervento per permettere ai partecipanti di seguirlo.
<b>Pre-condizioni</b>	L'utente deve aver aperto la pagina dell'intervento che desidera avviare.
<b>Post-condizioni per Successo</b>	La stanza virtuale è stata aperta e i partecipanti possono accedere all'evento.
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone “Apri stanza”.
<b>Attori primari</b>	Organizzatore dell'evento.

### **Scenario o (default):**

1. L'utente clicca sul bottone “Apri stanza”.
  - a. Il sistema apre la stanza virtuale e mostra i comandi per avviare il riconoscimento vocale.
2. L'utente avvia il riconoscimento vocale.

- a. Il sistema chiede il permesso di utilizzare il dispositivo di input audio predefinito.
3. L'utente conferma.
- a. Il sistema inizia il riconoscimento del parlato e disabilita la chiusura della stanza.
4. L'utente espone il discorso.
- a. Il sistema trascrive il parlato.
5. L'utente arresta il riconoscimento vocale.
- a. Il sistema ferma il servizio di riconoscimento e abilita la chiusura della stanza.
6. L'utente clicca sul bottone “Salva e Chiudi”
- a. Il sistema salva il testo trascritto nel database e chiude la stanza

#### **Scenario 1:**

6. L'utente clicca sul bottone “Chiudi stanza”.
- a. Il sistema chiede se si vuole salvare il testo prima di chiudere la stanza o meno.
7. L'utente sceglie di salvare il testo.
- a. Il sistema salva il testo trascritto nel database e chiude la stanza.

#### **Scenario 2:**

6. L'utente clicca sul bottone “Chiudi stanza”.
- a. Il sistema chiede se si vuole salvare il testo prima di chiudere la stanza o meno.
7. L'utente sceglie di non salvare il testo.
- a. Il sistema chiude la stanza.

Le risposte del sistema sono risultate conformi alle aspettative, l'interazione è andata come previsto, per cui il test può essere definito fallito (nessun errore riscontrato).

### **10.3. Ascolto - Rilettura Eventi**

L'ascolto di un evento o per meglio dire la partecipazione ad esso, comprende l'entrata in una stanza virtuale in cui è in corso l'evento stesso (in particolare un intervento) per leggere in tempo reale quello che viene trascritto dal sistema di riconoscimento. Il partecipante viene inserito tra i client del server socket aperto dal relatore e riceve quindi tutti i messaggi che vengono inviati. Il sistema è stato predisposto anche per la rilettura di tutti gli interventi effettuati durante un evento in un secondo momento. Di seguito vengono illustrati i cambiamenti effettuati al sistema per supportare questa funzione che va a completare il core del sistema stesso.

## Database

Il database ha subito una singola variazione, come per gli eventi sotto osservazione da parte di un utente, è stata inserita la lista di eventi a cui un utente partecipa, in modo da poter essere ritrovati facilmente in seguito per la rilettura degli interventi.

Al documento “Utente” è stato quindi aggiunto il campo seguente:

- **joinedEvent**: array di stringhe che contiene i riferimenti (ObjectID) agli eventi che l’utente ha seguito.

## Server Node

Nel server Node sono state incluse le API per l’aggiunta e la rimozione di un evento nella lista di quelli a cui un utente ha partecipato. Se un utente non vuole più vedere un determinato evento, infatti, può rimuoverlo per mettere ordine tra gli eventi a cui è più interessato e che vuole mantenere a portata di mano o, in questo caso, di click.

### user.js

allo schema dell’utente è stata aggiunta la lista degli eventi a cui ha partecipato, proprio come è stato fatto in precedenza con gli eventi osservati.

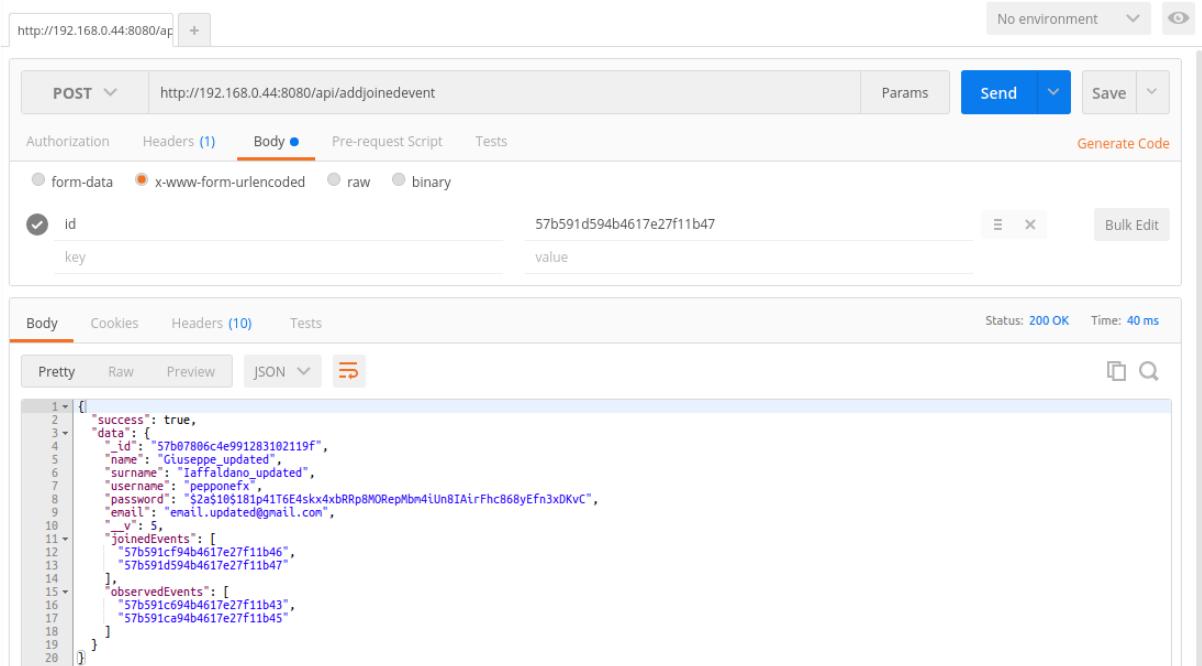
```
[...attributi dello schema...]
joinedEvents: {
    type: Array,
    required: false
}
[...eventuali altri attributi...]
```

### server.js

Le API eseguono le operazioni di aggiornamento della lista degli eventi a cui l’utente ha partecipato inserendo o eliminando il riferimento ad uno di essi. **/addjoinedevent** in particolare aggiunge il riferimento dell’evento, passato come parametro nel body della richiesta, alla lista di eventi a cui l’utente ha partecipato; **/removejoinedevent** invece rimuove dalla stessa lista il riferimento all’evento. L’ultima API (**/joinedevents**) infine, recupera dal database tutti gli eventi a cui l’utente a cui si fa riferimento ha partecipato e li restituisce come array.

Come al solito, sono stati effettuati dei test sulle API tramite Postman e verifiche sulla scrittura dei dati tramite Robomongo.

- Test di Aggiunta della Partecipazione ad un Evento

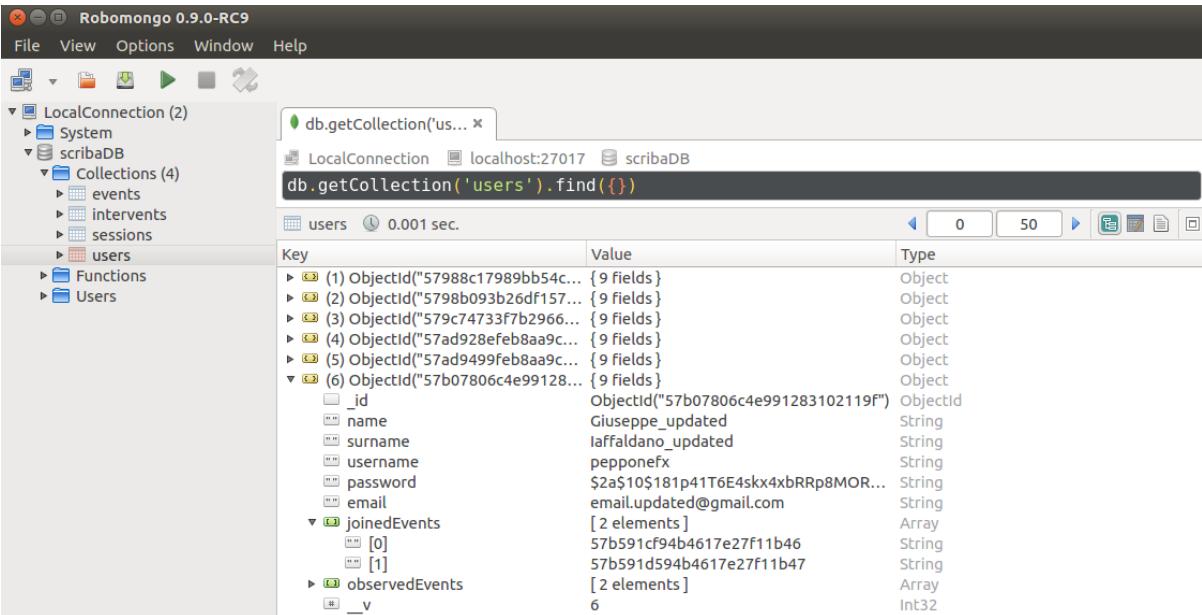


The screenshot shows a POST request to `http://192.168.0.44:8080/api/addjoinedevent`. The request body contains a single key-value pair: `id: 57b591d594b4617e27f11b47`. The response status is 200 OK with a time of 40 ms. The JSON response is:

```

1  {
2   "success": true,
3   "data": {
4     "_id": "57b07806c4e991283102119f",
5     "name": "Giuseppe_updated",
6     "surname": "Iaffaldano_updated",
7     "username": "pepponefx",
8     "password": "$2a$10$181p41T6E4skx4xbRRp8MORpMb4iUn8IAirFhcB68yEfn3xDKvC",
9     "email": "email.updated@gmail.com",
10    "__v": 5,
11    "joinedEvents": [
12      "57b591cf94b4617e27f11b46",
13      "57b591d594b4617e27f11b47"
14    ],
15    "observedEvents": [
16      "57b591c694b4617e27f11b43",
17      "57b591ca94b4617e27f11b45"
18    ]
19  }
20 }

```



The screenshot shows the Robomongo interface connected to a MongoDB database named `scribaDB`. The `users` collection is selected. A query `db.getCollection('users').find({})` is run, returning the following results:

Key	Value	Type
<code>_id</code>	<code>ObjectId("57b07806c4e991283102119f")</code>	<code>Object</code>
<code>name</code>	<code>Giuseppe_updated</code>	<code>String</code>
<code>surname</code>	<code>Iaffaldano_updated</code>	<code>String</code>
<code>username</code>	<code>pepponefx</code>	<code>String</code>
<code>password</code>	<code>\$2a\$10\$181p41T6E4skx4xbRRp8MORpMb4iUn8IAirFhcB68yEfn3xDKvC</code>	<code>String</code>
<code>email</code>	<code>email.updated@gmail.com</code>	<code>String</code>
<code>joinedEvents</code>	<code>[2 elements]</code>	<code>Array</code>
<code>joinedEvents[0]</code>	<code>57b591cf94b4617e27f11b46</code>	<code>String</code>
<code>joinedEvents[1]</code>	<code>57b591d594b4617e27f11b47</code>	<code>String</code>
<code>observedEvents</code>	<code>[2 elements]</code>	<code>Array</code>
<code>__v</code>	<code>6</code>	<code>Int32</code>

- Test di Lettura della lista di Eventi a cui un utente ha partecipato

http://192.168.0.44:8080/api/joinedevents

GET

Headers (1)

Params

Send

Save

Generate Code

Authorization

key: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI1N2lw...

Body

Cookies

Headers (10)

Tests

Pretty

Raw

Preview

JSON

```

1  [
2   "success": true,
3   "data": [
4     {
5       "id": "57b591cf94b4617e27f11b46",
6       "title": "evento di test 4",
7       "startDate": "2016-10-15T15:30:00.000Z",
8       "endDate": "2016-10-15T18:30:00.000Z",
9       "location": "Uniba - Via Orabona, 4",
10      "status": "programmed",
11      "organizer": "pepponefx",
12      "public": true,
13      "v": 0,
14      "allowed": []
15    },
16    {
17      "id": "57b591d594b4617e27f11b47",
18      "title": "evento di test 5",
19      "startDate": "2016-10-15T15:30:00.000Z",
20      "endDate": "2016-10-15T18:30:00.000Z",
21      "location": "Uniba - Via Orabona, 4",
22      "status": "programmed",
23      "organizer": "pepponefx",
24      "public": true,
25      "v": 0,
26      "allowed": []
27    }
28  ]
29

```

Status: 200 OK Time: 45 ms

- Test di Eliminazione della Partecipazione ad un Evento

http://192.168.0.44:8080/api/removejoinedevent

POST

Headers (1)

Body

Params

Send

Save

Generate Code

Authorization

form-data

x-www-form-urlencoded

raw

binary

id: 57b591d594b4617e27f11b47

Body

Cookies

Headers (10)

Tests

Pretty

Raw

Preview

JSON

```

1  {
2   "success": true,
3   "data": {
4     "_id": "57b591d594b4617e27f11b47",
5     "name": "Giuseppe_updated",
6     "surname": "Iaffaldano_updated",
7     "username": "pepponefx",
8     "password": "$2as10$181p41T6E4skx4xbRRp8M0RepMb4iUn8IAirFhc868yEfFn3xDKvC",
9     "email": "email.updated@mail.com",
10    "v": 6,
11    "joinedEvents": [
12      "57b591cf94b4617e27f11b46"
13    ],
14    "observedEvents": [
15      "57b591c694b4617e27f11b43",
16      "57b591ca94b4617e27f11b45"
17    ]
18  }
19

```

Status: 200 OK Time: 22 ms

Key	Value	Type
1 ObjectId("57988c17989bb54c...")	{ 9 fields }	Object
2 ObjectId("5798b093b26df157...")	{ 9 fields }	Object
3 ObjectId("579c74733f7b2966...")	{ 9 fields }	Object
4 ObjectId("57ad928efeb8aa9c...")	{ 9 fields }	Object
5 ObjectId("57ad9499feeb8aa9c...")	{ 9 fields }	Object
6 ObjectId("57b07806c4e99128...")	{ 9 fields }	Object
_id	ObjectId("57b07806c4e991283102119f")	Objectid
name	Giuseppe_updated	String
surname	Iaffaldano_updated	String
username	pepponefx	String
password	\$2a\$10\$181p41T6E4skx4xbRRp8MOR...	String
email	email.updated@gmail.com	String
joinedEvents	[ 1 element ]	Array
[ 0 ]	57b591cf94b4617e27f11b46	String
observedEvents	[ 2 elements ]	Array
__v	7	Int32

## Server Ionic

Sul lato client l'operazione di inserimento di un evento tra quelli a cui l'utente ha partecipato avviene in maniera trasparente, è stata esposta invece la nuova pagina creata, ossia quella contenente la lista degli eventi seguiti nella quale l'utente potrà rileggere gli interventi di ciascun evento o rimuovere eventi a cui non è più interessato.

Sono stati aggiunti inoltre i servizi di comunicazione con il server Node nelle apposite collezioni.

### personal-home-page.html - observed-events-page.html

In queste due pagine sono stati aggiunti i bottoni per l'avvio dell'ascolto dell'intervento in corso nella lista che visualizza gli eventi disponibili. Il bottone sarà visibile all'utente solo dal momento in cui viene aperta la stanza virtuale per permettere l'accesso ai partecipanti.

```
<button *ngIf="event.status=='ongoing'" (click)="openEvent(event)">
  <ion-icon name="play"> </ion-icon>
</button>
```

### personal-home-page.ts - observed-events-page.ts

Lo script è stato arricchito del metodo per aprire l'ascolto dell'intervento, in particolare il metodo cerca (all'interno dell'intervento selezionato) la sessione in corso e di tale sessione l'evento in corso, passa poi i riferimenti alla pagina di ascolto dell'evento su cui l'utente viene indirizzato e inserisce l'evento nella lista di quelli a cui l'utente ha partecipato.

```

openEvent(eventToOpen){
  this.us.addJoinedEvent(eventToOpen._id).map(res=>res.json()).subscribe(data=>{
    alert(data.msg);
  });
  this.events.forEach(event => {
    if(event._id==eventToOpen._id){
      event.sessions.forEach(session => {
        if(session.status=="ongoing"){
          session.intervents.forEach(intervent => {
            if(intervent.status=="ongoing"){
              this.nav.push(ListenInterventPage,{intervent: intervent})
            }
          });
        }
      });
    }
  });
}

```

#### joined-events-page/joined-events-page.html - joined-events-page.ts

La pagina mostra la lista degli eventi che l'utente ha seguito, allo stesso modo delle altre liste, in un momento successivo all'evento, l'utente può espandere i sottomenù ad albero per poter rileggere gli interventi effettuati (il cui testo è stato memorizzato nel database) durante tutto l'evento.

Lo script recupera tutte le informazioni degli eventi seguiti dall'utente autenticato sfruttando l'apposito servizio aggiunto alla collezione; predispone quindi la lista degli eventi che verrà visualizzata nella pagina html.

#### listen-intervent-page/listen-intervent-page.html - listen-intervent-page.ts

La pagina rappresenta la stanza virtuale a cui gli utenti accedono per seguire l'intervento in corso e in cui lo speaker sta trascrivendo (tramite il servizio di riconoscimento) ciò che proferisce.

Lo script controlla la comunicazione con il server socket.io da cui riceve i messaggi che visualizza nella pagina html e a cui invia le richieste di connessione o disconnessione del singolo partecipante quando necessario.

```

import { Component } from '@angular/core';
import {NavController, NavParams} from 'ionic-angular';
import {User} from '../../services/models/user-model';
import {Intervent} from '../../services/models/intervent-model';

declare var io: any;
@Component({
  templateUrl: 'build/pages/listen-intervent-page/listen-intervent-page.html',

```

```

})
export class ListenInterventPage {
    private localUser=JSON.parse(window.localStorage.getItem("user"));
    private user= new User(this.localUser.name, this.localUser.surname,
this.localUser.username, this.localUser.password, this.localUser.email);
    private intervent = this.np.get('intervent');
    private room = null;
    constructor(private np: NavParams, private nav: NavController) {
        this.room = io.connect('http://192.168.0.44:'+this.intervent.port);
        this.room.emit('client_type', {text: "Listener"});

        this.room.on('previous_text', (data) => {
            document.getElementById('text').innerHTML += data.text;
        })

        this.room.on('server_message', (data) => {
            document.getElementById('text').innerHTML += data.text;
        });

        this.room.on('closing_room', (data) => {
            this.room=null;
            alert(data.text);
        });
    }
    close() {
        this.room.disconnect();
        this.nav.pop();
    }
}

```

services/models/user-model.ts

Al modello dell'utente è stato aggiunto l'attributo che contiene la lista di eventi seguiti, esattamente come è stato fatto per gli eventi osservati.

```

[...]
public joinedEvents? :string
[...]

```

services/user-services.ts

Alla collezione di servizi per l'utente sono stati aggiunti quelli per l'inserimento e la rimozione di un intervento nella lista di quelli a cui l'utente ha partecipato che restituiscono al chiamante il solito oggetto Observable contenente le risposte che giungono dal server.

services/event-services.ts

La collezione di servizi relativi alla gestione degli eventi è stata completata con i metodi che preparano la richiesta della lista di eventi a cui l'utente autenticato ha partecipato, restituendola come Observable al chiamante.

## Test

Come per le altre user story, anche per il completamento di quest'ultima sono stati effettuati dei test di simulazione dell'interazione di un utente secondo lo schema seguente, per verificarne il giusto comportamento

<b>Caso d'uso:</b> Ascoltare Intervento	
<b>Descrizione</b>	L'utente entra nella stanza virtuale dove è in corso un intervento per ascoltarlo.
<b>Pre-condizioni</b>	L'intervento deve essere stato aperto; l'utente deve aver aperto la pagina degli eventi disponibili ("Tutti gli Eventi") oppure la pagina degli eventi osservati ("Eventi Osservati") dal menù laterale.
<b>Post-condizioni per Successo</b>	L'utente viene indirizzato nella pagina per l'ascolto dell'intervento (stanza virtuale).
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone "Ascolta".
<b>Attori primari</b>	Qualsiasi utente.

### Scenario o (default):

1. L'utente clicca sul bottone "Ascolta".
  - a. Il sistema aggiunge l'evento alla lista di quelli a cui l'utente ha partecipato, apre la pagina della stanza virtuale e mostra il testo trascritto durante l'intervento.
2. L'utente clicca sul bottone "Chiudi"
  - a. Il sistema disconnette l'utente e lo reindirizza alla pagina iniziale.

<b>Caso d'uso:</b> Rileggere Evento	
<b>Descrizione</b>	L'utente rilegge il testo trascritto degli interventi di un evento.
<b>Pre-condizioni</b>	L'evento è completato del tutto o parzialmente; l'utente deve aver aperto la pagina degli eventi disponibili ("Tutti gli Eventi") oppure la pagina degli eventi osservati ("Eventi Osservati") oppure la pagina degli eventi seguiti ("Eventi Seguiti") dal menù laterale.
<b>Post-condizioni per Successo</b>	L'utente visualizza il testo trascritto durante l'evento.
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente espande il sottomenù ad albero di un evento fin dove è interessato.
<b>Attori primari</b>	Qualsiasi utente.

**Scenario o (default):**

1. L'utente espande il sottomenù ad albero di un evento fin dove è interessato.
  - a. Il sistema estende l'albero mostrando il testo di ciascun intervento espando (ove presente).

<b>Caso d'uso:</b> Rimuovere Evento Seguito	
<b>Descrizione</b>	L'utente rimuove un evento dalla lista di eventi seguiti.
<b>Pre-condizioni</b>	L'utente deve aver aperto la pagina degli eventi seguiti ("Eventi Seguiti") dal menù laterale.
<b>Post-condizioni per Successo</b>	L'evento è stato rimosso dalla lista di quelli seguiti.
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.
<b>Evento innescante</b>	L'utente clicca il bottone "Elimina".
<b>Attori primari</b>	Qualsiasi utente.

**Scenario o (default):**

1. L'utente clicca sul bottone "Elimina" di un evento.
  - a. Il sistema chiede conferma.

2. L'utente conferma.

- a. Il sistema rimuove l'evento dalla lista di quelli seguiti e aggiorna la pagina.

Scenario 1:

2. L'utente NON conferma.

- a. Il sistema NON apporta modifiche alla lista degli eventi seguiti.

Dopo aver eseguito le azioni previste, il sistema ha risposto esattamente come ci si aspettava, per questo motivo il test si è potuto considerare fallito (nessun errore riscontrato).

## 10.4. Interventi dei partecipanti – Domande al relatore

Una ulteriore issue implementata durante questo sprint è stata quella che prevede l'interazione del partecipante che pone domande ai relatori. Un partecipante normodotato, tipicamente alzerebbe la mano per richiedere la parola e formulerebbe verbalmente la propria domanda. Soggetti audiolesi però, sono spesso inibiti nel compiere questa operazione poichè potrebbero aver problemi nel modulare bene il tono di voce e le parole, problema frequente in individui definiti sordo-muti. Per questo motivo e per tenere traccia nel sistema delle domande effettuate durante un intervento, si fornisce la funzionalità che permette di alzare la mano virtualmente e porre una domanda scritta al relatore che la visualizzerà sulla sua interfaccia Web Desktop.

### 10.4.1. Database

Sul database MongoDB è stato aggiunto al modello dell'intervento un array di domande che sarà riempito durante l'intervento con una serie di oggetti composti dagli attributi “user” e “text” che rispettivamente indicano lo username di chi pone la domanda e il testo della domanda stessa.

### 10.4.2. Server Node

Dal punto di vista del server Node, invece, è stato aggiunto il protocollo di gestione delle domande, più precisamente al server Socket.IO che gestisce l'esecuzione dell'intervento. In particolare, quando un partecipante pone una domanda, la socket in ascolto riceve la domanda, la memorizza localmente e la reinvia a tutti i partecipanti compreso il relatore in modo che possano leggerla. L'operazione è eseguita esattamente come se si stesse inoltrando una trascrizione, utilizzando quindi il broadcast.

L'unica API aggiunta (**/addquestion**) si occupa di aggiungere all'array contenente le domande di un intervento, la domanda passata come parametro nel body della richiesta, assieme al riferimento dell'intervento a cui la domanda si riferisce.

L'API aggiunta è stata testata attraverso la simulazione di una query con Postman, è stato poi verificata la corretta esecuzione mediante RoboMongo.

- Test di aggiunta di una domanda

```

1  {
2     "success": true,
3     "data": {
4         "_id": "57b6b80f934bf5c81c518a02",
5         "title": "Intervento di test",
6         "date": "2016-10-15T16:00:00.000Z",
7         "duration": 60,
8         "speaker": "speaker1",
9         "session": "57b6b7fe934bf5c81c518a01",
10        "status": "programmed",
11        "__v": 0,
12        "port": 41490,
13        "text": "Testo di esempio da salvare nell'intervento per verificare che l'API funzioni correttamente.",
14        "questions": [
15            {
16                "user": "utente",
17                "text": "Questa è una domanda?"
18            }
19        ]
20    }
21 }

```

**Robomongo 0.9.0-RC9**

File View Options Window Help

LocalConnection (2)

- System
- scribaDB
  - Collections (4)
    - events
    - intervents
    - sessions
    - users
- Functions
- Users

db.getCollection('int...')

LocalConnection localhost:27017 scribaDB

db.getCollection('intervents').find({})

Key	Value	Type
» (1) ObjectId("57988cc6989bb54...")	{ 10 fields }	Object
» (2) ObjectId("57988cdb989bb54...")	{ 8 fields }	Object
» (3) ObjectId("57b6b80f934bf5c8...")	{ 11 fields }	Object
__id	ObjectId("57b6b80f934bf5c81c518a02")	Objectid
title	Intervento di test	String
date	2016-10-15 16:00:00.000Z	Date
duration	60	Int32
speaker	speaker1	String
session	57b6b7fe934bf5c81c518a01	String
status	programmed	String
__v	1	Int32
port	41490	Int32
text	Testo di esempio da salvare nell'interv... [ 1 element ] { 2 fields }	String
questions	[ 1 element ] { 2 fields }	Array
[0]	Questa è una domanda? utente	Object
text	Questa è una domanda?	String
user	utente	String

#### 10.4.3. Server Ionic

Sul lato Ionic sono state effettuate poche modifiche all'aspetto grafico, l'intervento maggiore è stato il completamento del protocollo di comunicazione sulla socket dal lato client.

listen-intervent-page/listen-intervent-page.html - listen-intervent-page.ts

Alla pagina che gestisce l'ascolto di un intervento, esposta quindi sul lato mobile, è stato aggiunto un bottone rappresentante una mano, ad indicare proprio il gesto di alzata di mano,

il bottone espande un form (grazie alla direttiva IF di angular) che gestisce l'inserimento di una domanda. Infine è stata aggiunta la lista di domande effettuate durante l'intervento al di sotto della trascrizione grazie alla direttiva FOR.

Lo script che controlla la pagina contiene invece la logica di invio della domanda attraverso la socket, in particolare invia una "question" al server socket.IO contenente lo username di chi pone la domanda e il testo della domanda stessa. Inoltre gestisce l'array contenente le domande effettuate durante l'intervento e legato alla lista visualizzata nella pagina.

intervent-page/intervent-page.html - intervent-page.ts

Alla pagina dedicata allo speaker è stata aggiunta la sola lista delle domande che arrivano dai partecipanti e che vengono visualizzate al di sotto della trascrizione.

Lo script controlla invece l'array che contiene le domande che viene riempito ogni qualvolta si riceve dal server una nuova "question".

Al termine dell'intervento, quando lo speaker vorrà salvare il testo, sarà inviata richiesta, al servizio che gestisce l'interazione con il server per le operazioni inerenti agli eventi, di salvataggio di tutte le domande ricevute.

services/event-services.ts

Il servizio è stato completato con il metodo che richiede al server Node di salvare una domanda per un determinato intervento; tali dati sono inglobati nel body della richiesta e la risposta viene restituita al chiamante tramite il solito oggetto Observable.

#### 10.4.4. Test

Anche per questa issue, sono state effettuate le consuete simulazioni dell'interazione dell'utente, prima della sua chiusura, secondo il modello che segue.

<b>Caso d'uso:</b> Porre Domanda	
<b>Descrizione</b>	L'utente pone una domanda al relatore attraverso il sistema mentre sta ascoltando un intervento.
<b>Pre-condizioni</b>	L'intervento deve essere stato avviato; l'utente deve essere in ascolto.
<b>Post-condizioni per Successo</b>	La domanda viene inviata a tutti i partecipanti compreso lo speaker.
<b>Post-condizioni per Fallimento</b>	Viene mostrato un messaggio di errore esplicativo del problema.

<b>Evento innescante</b>	L'utente clicca il bottone "Alza la Mano".
<b>Attori primari</b>	Partecipante.

**Scenario o (default):**

1. L'utente clicca sul bottone "Alza la Mano".
  - a. Il sistema mostra il campo per l'inserimento della domanda e il bottone per l'invio (inizialmente disattivato).
2. L'utente inserisce la domanda.
  - a. Il sistema attiva il bottone "Invia".
3. L'utente clicca il bottone "Invia".
  - a. Il sistema invia la domanda a tutti i partecipanti e allo speaker.

**Scenario 1:**

3. L'utente cancella la domanda.
  - a. Il sistema disattiva il bottone "Invia".

La risposta del sistema è risultata conforme alle aspettative, per questo il test può essere definito fallito (nessun errore riscontrato).

# 11. Sprint 6 - Deployment multi-piattaforma

Durante il meeting relativo allo sprint precedente, sono stati analizzati i progressi effettuati e ripetuti i vari test, le modifiche sono state accettate previa creazione di nuove issue di perfezionamento del sistema etichettate come enhancement.

Per lo sprint che sarà descritto in questo capitolo, sono state invece programmate (e inserite nello sprint backlog) le issue di deploy sulle varie piattaforme che si prevede di utilizzare: l'ambiente desktop con browser Chrome, l'ambiente mobile Android e l'ambiente mobile iOS. Prima di procedere con il deployment del server Ionic sulle diverse piattaforme, il database e il server Node sono stati spostati su una macchina che fungerà da server provvisorio. Di seguito si ripercorrerà la strada intrapresa per il completamento dei task previsti.

## 11.1. Deployment del server Node

Durante l'intero sviluppo del sistema, il codice è stato ospitato e testato sulla macchina windows dello sviluppatore. Per verificare che tutto funzioni correttamente in un contesto reale, è stato simulato tale contesto spostando database e lato Node su una macchina adibita a Server. Il laboratorio di ricerca Collab che ha ospitato lo sviluppatore durante il lavoro di tesi, dispone di una macchina con processore con architettura a 32 bit e sistema operativo Debian. Il problema è nato quando ci si è resi conto che MongoDB come Angular 2 offre i pacchetti di installazione per sistemi ad architettura a 64 bit. Per un primo test di questo tipo quindi, lo sviluppatore ha messo a disposizione un laptop HP Pavilion g6 che monta un processore Intel core i5-2450M @ 2.5GHz e 8 GB di ram DDR3 su cui è stato installato il sistema operativo debian-based Ubuntu alla versione 16.04 LTS. Successivamente è stato assegnato alla macchina un indirizzo IP fisso per la rete locale: 192.168.0.44. Questo ha permesso allo sviluppatore di installare tutta l'infrastruttura necessaria all'esecuzione del sistema come sarà descritto di seguito.

### 11.1.1. Installazione del database MongoDB

Per l'installazione del database non è bastato utilizzare il semplice comando apt-get, è stato infatti necessario aggiungere il repository di MongoDB (`sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927`), creare la lista dei file per il database (`echo "deb http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list`) e aggiornare la lista dei pacchetti (`sudo apt-get update`).

Al termine della procedura, è stato installato il database attraverso il comando `sudo apt-get install -y mongodb-org`; creato il file di configurazione “`mongodb.service`” nella directory “`/etc/systemd/system/`” contenente il seguente codice:

```
[Unit]
Description = High-performance, schema-free document-oriented database
After = network.target

[Service]
User = mongodb
ExecStart = /usr/bin/mongod --quiet --config /etc/mongod.conf

[Install]
WantedBy = multi-user.target
```

- **Unit**: contiene una panoramica così come le dipendenze che devono essere soddisfatte prima di avviare il servizio.
- **Service**: definisce come deve essere avviato il servizio. La direttiva User specifica che il server verrà eseguito sotto l'utente “`mongodb`”, e la direttiva ExecStart definisce il comando di avvio per il server MongoDB .
- **Install**: comunica a “`systemd`” quando il servizio deve essere avviato automaticamente. Il `multi-user.target` è una sequenza standard di avvio del sistema, il server verrà avviato automaticamente durante l'avvio del sistema operativo.

Il passo successivo è stato avviare il servizio appena creato con `systemctl` (`sudo systemctl start mongodb`).

A questo punto è stato installato anche il tool Robomongo per verificare che il database fosse davvero funzionante ed è stato constatato che tutto procedeva nel verso giusto.

### 11.1.2. Installazione di NodeJS

L'installazione di Node tramite `apt-get`, permette di ottenere la versione `0.10.x` per Ubuntu, per ottenere invece la versione `6.x` necessaria al sistema, c'è stato bisogno di un PPA (personal package archive) al fine di ottenere l'accesso ai suoi contenuti. Nella home directory, tramite `curl` per recuperare lo script di installazione per la versione preferita, è stato lanciato da shell il comando `curl -sL https://deb.nodesource.com/setup_6.x -o nodesource_setup.sh`; dopodichè è stato lanciato lo script (`sudo bash nodesource_setup.sh`) che aggiorna automaticamente i pacchetti `apt`, dopodichè è stato normalmente installato Node tramite `sudo apt-get install nodejs` e configurata la build (`sudo apt-get install build-essential`).

### 11.1.3. Installazione del server realizzato

Il server è stato trasferito sulla macchina creando la cartella “git” sotto “home” e lanciando la clonazione del repository. Una volta scaricato il codice, è bastato installare tutte le dipendenze tramite il comando `npm install` e lanciato il server (`node server.js`) dopo aver modificato l’indirizzo IP nel codice per essere corrispondente a quello assegnato alla macchina.

## 11.2. Deployment del server Ionic

A questo punto, installato il server Node su una macchina è possibile effettuare il deploy del lato Ionic su qualsiasi altra macchina per servire qualsiasi piattaforma Desktop tramite browser Google Chrome. Come vedremo, inoltre, grazie alle potenzialità offerte da Ionic è facile costruire i file di installazione dell’applicazione Android e iOS per essere distribuiti agli utenti.

### 11.2.1. Applicazione Web Desktop

Sulla stessa macchina del server Node (per motivi di irreperibilità di un’altra macchina), è stato installato il framework Ionic tramite `npm install -g ionic@beta` proprio come è stato fatto in precedenza su Windows. Terminata la procedura, considerato che il codice era già presente sulla macchina dopo la clonazione del repository, è bastato installare le dipendenze anche per questo progetto (`npm install`), modificare l’indirizzo di riferimento al server per l’accesso alle API e avviare il server Ionic su un indirizzo accessibile dall’esterno (`ionic serve --address 192.168.0.44`). Nel caso specifico l’indirizzo sarà accessibile dai soli utenti connessi alla stessa rete locale su cui è in esecuzione il server; successivamente basterà utilizzare un indirizzo IP pubblico per consentire l’accesso a tutti.

### 11.2.2. Applicazione Android

Il deploy del sistema Ionic su Android è una procedura davvero molto semplice grazie all’integrazione del framework con Cordova. Ionic infatti permette di utilizzare un emulatore Android per il test oppure collegare il dispositivo su cui deve essere abilitato il debug USB per lanciare l’applicazione sul dispositivo stesso. Un’altra alternativa è quella di creare una build quindi un file APK installabile su un qualsiasi dispositivo previa accettazione di fonti sconosciute.

La prima cosa da fare in tutti i casi è comunicare al sistema che si intende sviluppare anche per Android, per far ciò basta lanciare il comando `ionic platforms add android` da console.

A questo punto il sistema è pronto per essere deployato sul dispositivo. Per il semplice lancio dell'applicazione sul dispositivo basta collegare lo smartphone al PC, abilitare l'opzione di debug USB e lanciare il comando `ionic run android`.

Per creare il file APK pronto alla distribuzione, bisogna innanzitutto disabilitare la console di debug (`cordova plugin rm cordova-plugin-console`); creare un file APK senza firma attraverso il comando `ionic build --release android`. A questo punto il file è già installabile sul dispositivo. Per poterla pubblicare, l'app ha bisogno di essere firmata, tramite il tool “keytool” è possibile richiedere una chiave per la firma:

```
keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA  
-keysize 2048 -validity 10000
```

Successivamente è necessario firmare l'APK generato tramite il tool “jarsigner”:

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-  
key.keystore Scriba-release-unsigned.apk Scriba
```

A questo punto è necessario allineare lo zip per ottimizzare l'APK attraverso l'apposito tool in bundle: `zipalign -v 4 Scriba-release-unsigned.apk Scriba.apk`.

Il file ottenuto è ora pronto per essere pubblicato sugli store.

Per quanto riguarda questo progetto, la procedura è stata arrestata alla fase in cui è stato creato il file APK senza firma, in modo da poter essere prima testato sul dispositivo dello sviluppatore e successivamente rilasciato sotto accettazione del team di progetto.

### 11.2.3. Applicazione iOS

Per effettuare il deployment su iOS, è stato strettamente necessario recuperare uno smartphone Apple e una macchina Mac. Una volta ottenuti un iPhone 5S e un Mac Mini, è stato installato Node come in precedenza, Ionic e tutte le dipendenze necessarie per importare il progetto su MacOS. Alla fine della procedura di importazione nella macchina del codice di progetto, è stata aggiunta la piattaforma iOS al progetto come per Android con il comando `Ionic platforms add ios`. Al termine è stata costruita la build per iOS tramite `Ionic build ios`. Ottenuta la build, nei file di progetto sarà stato creato un file di progetto importabile in XCode. Aperto XCode e importato il file, è stato necessario ottenere un account da sviluppatore Apple che è stato fornito da Informatici Senza Frontiere. Ora che tutto il necessario è pronto, è bastato selezionare su XCode il dispositivo su cui si vuole effettuare il deploy e cliccare sul tasto “run”. A questo punto viene costruita e deployata l'app sul dispositivo ed è possibile utilizzarla. Al termine dello sviluppo e dei test, sarà pubblicata l'app sullo store di casa Apple.

### 11.3. Predisposizione per il deployment esterno

Fino a questo momento il sistema ha sempre lavorato in locale, per questo le porte TCP utilizzate non erano vincolate e sono state sempre accessibili. Quando si vuole portare il sistema sul server esterno, però, il problema da superare è l'apertura delle porte TCP necessarie all'accesso dall'esterno al sistema. All'avvio di un intervento, viene avviato l'ascolto delle socket su una porta libera, per la sicurezza del sistema e per la necessità di non occupare le "well known ports" su cui comunicano alcuni sistemi e giochi online, è impensabile aprire tutte le porte disponibili. Per ovviare a questo problema, la comunicazione sulla socket quindi l'intero server Socket.IO è stato ristrutturato per comunicare su una sola porta e gestire interventi contemporanei ma separati logicamente attraverso le "room".

Una "room" definisce uno spazio logico riservato alle sole socket che il server decide di aggiungere alla stessa, tutte le socket in tutte le stanze sono comunque gestite globalmente poiché connesse allo stesso server. Di seguito viene riportato il nuovo codice ristrutturato con le dovute spiegazioni del caso.

```
//DEPENDENCIES
var server = require("http").Server(express);
var io = require("socket.io")(server);
server.listen(8081);

var allClients=[];
var openRooms = {};
var unsavedQuestions = {};
io.on('connection', function (socket) {
    allClients.push(socket);

    socket.on('client_type',function(data){
        var msg="User Type: ";
        msg += data.text;
        console.log(msg);
    });

    //GESTIONE RELATORI
    socket.on('open_room', function(data){
        socket.join(data.room);
        openRooms[data.room]="";
        unsavedQuestions[data.room]=[];
    });

    socket.on('close_room', function(data){
        socket.broadcast.to(data.room).emit('disconnection', {text: "La stanza è stata
chiusa!"})
        socket.leave(data.room);
        //RIAGGIORNA LO STATO DI INTERVENTO; SESSIONE ED EVENTO
        Intervent.findOne({ _id: data.room }, function (err, intervent){
            if(err) throw err;

            intervent.status= 'programmed';
        });
    });
});
```

```

intervent.save();

Session.findOne({ _id: intervent.session}, function (err, session){
    if(err) throw err;

    session.status= 'programmed';
    session.save();

    Event.findOne({ _id: session.event }, function (err, event){
        if(err) throw err;
        event.status= 'programmed';
        event.save();
    });
});
});

socket.on('speaker_message',function(data){
    openRooms[data.room] += data.text + " ";
    socket.broadcast.to(data.room).emit('new_transcription', {text:data.text});
});

//GESTIONE PARTECIPANTI
socket.on('join_room', function(data){
    socket.join(data.room);
    if(data.type=='Listener'){
        socket.emit('previous_text', {text: openRooms[data.room], questions:
unsavedQuestions[data.room]});
    }
});

socket.on('leave_room', function(data){
    socket.leave(data.room);
});

socket.on('client_question',function(data){
    unsavedQuestions[data.room].push({text:data.text, user: data.user});
    socket.broadcast.to(data.room).emit('question', {text:data.text, user: data.user});
});
});

```

- `io.on('connection')`: Alla connessione di un utente generico al sistema, questo viene inserito subito nell'array contenente tutti gli utenti connessi per tutte le stanze.
- `socket.on('client_type')`: Alla ricezione del tipo di utente che è connesso viene stampato nel log un messaggio che lo notifica.
- `socket.on('open_room')`: Gestisce la richiesta di apertura di una stanza (possibile solo da parte di uno “Speaker”), viene modificato lo stato dell'intervento in “ongoing” in modo che i partecipanti possano accedervi e viene creata una stanza denominata attraverso l'identificativo univoco dell'intervento afferente.
- `socket.on('close_room')`: Gestisce la richiesta di chiusura della stanza, lo stato dell'intervento viene modificato nuovamente e viene inviata a tutti i client connessi alla stanza una notifica di chiusura dell'intervento.

- `socket.on('speaker_message')`: Gestisce la ricezione della trascrizione da parte dello speaker in maniera da reinviarla in broadcast a tutti gli utenti connessi alla stanza relativa all'intervento.
- `socket.on('join_room')`: Gestisce la richiesta di accesso di un partecipante ad un intervento, in particolare, il client viene inserito all'interno della stanza relativa all'intervento in questione. Se il client è di tipo "Listener" gli viene inviato tutto il testo trascritto e tutte le domande effettuate prima del suo accesso.
- `socket.on('leave_room')`: Gestisce la richiesta di uscita da un intervento, il sistema elimina il client dalla stanza che resta connesso al server finché non desidera chiudere definitivamente la connessione.
- `socket.on('client_question')`: Gestisce l'invio di domande da parte dei client "Listener", la domanda viene inviata a tutti i partecipanti ad un intervento e allo speaker.

## 11.4. Deployment esterno – Docker<sup>82</sup>

Docker è un progetto open-source che automatizza il deployment di applicazioni all'interno di container software, fornendo un'astrazione addizionale grazie alla virtualizzazione a livello di sistema operativo di Linux. Docker utilizza le funzionalità di isolamento delle risorse del kernel Linux per consentire a "container" indipendenti di coesistere sulla stessa istanza di Linux, evitando l'installazione e la manutenzione di una macchina virtuale.

Secondo la ditta di analisi industriale 451 Research,

"Docker è uno strumento che può pacchettizzare un'applicazione e le sue dipendenze in un container virtuale che può essere eseguito su qualsiasi server Linux."<sup>83</sup>

Docker implementa API di alto livello per gestire container che eseguono processi in ambienti isolati.<sup>84</sup> Poiché utilizza delle funzionalità del kernel Linux, un container di Docker, a differenza di una macchina virtuale, non include un sistema operativo separato. Al contrario, utilizza le funzionalità del kernel e sfrutta l'isolamento delle risorse (CPU, memoria, I/O a blocchi, rete) ed i namespace separati per isolare ciò che l'applicazione può vedere del sistema operativo. Docker accede alle funzionalità di virtualizzazione del kernel

---

<sup>82</sup> Docker: <https://www.docker.com/>

<sup>83</sup> Katherine Noyes, Docker: A 'Shipping Container' for Linux Code, Linux.com, 1º agosto 2013

<sup>84</sup> Abel Avram, Docker: Automated and Consistent Software Deployments, InfoQ, 27 marzo 2013

Linux o direttamente utilizzando la libreria libcontainer, che è disponibile da Docker 0.9, o indirettamente attraverso libvirt, LXC or systemd-nspawn.<sup>85</sup> <sup>86</sup>

Utilizzando i container, le risorse possono essere isolate, i servizi limitati ed i processi avviati in modo da avere una prospettiva completamente privata del sistema operativo, col loro proprio identificativo, file system ed interfaccia di rete. Più container condividono lo stesso kernel, ma ciascuno di essi può essere costretto ad utilizzare una certa quantità di risorse, come la CPU, la memoria e l'I/O.

L'utilizzo di Docker per creare e gestire i container può semplificare la creazione di sistemi distribuiti, permettendo a diverse applicazioni o processi di lavorare in modo autonomo sulla stessa macchina fisica o su diverse macchine virtuali. Ciò consente di effettuare il deployment di nuovi nodi solo quando necessario, permettendo uno stile di sviluppo del tipo platform as a service (PaaS). Docker inoltre semplifica la creazione e la gestione di code di lavori in sistemi distribuiti.<sup>87</sup> <sup>88</sup>

Docker può essere integrato in varie infrastrutture, tra cui Amazon Web Services, Ansible, CFEngine, Chef, Google Cloud Platform, IBM Bluemix, Jelastic, Jenkins, Microsoft Azure, NoMachine, OpenStack Nova, OpenSVC, Puppet, Salt, e Vagrant.

### Deployment di Scriba con Docker

Una volta installato Docker, come prima cosa sono stati creati tre Dockerfile rispettivamente per la creazione delle immagini dei servizi mongoDB; server Node; server Ionic.

Il Dockerfile serve per automatizzare la procedura di creazione e modifica di un container. Attraverso il Dockerfile infatti è possibile non solo fare il deploy istantaneo automatizzato di più istanze e più container, ma è anche possibile eseguire il provisioning di queste istanze, automatizzando task di gestione del sistema, installazione del software e tanto altro (in figura seguente la struttura in layer di un container Docker).

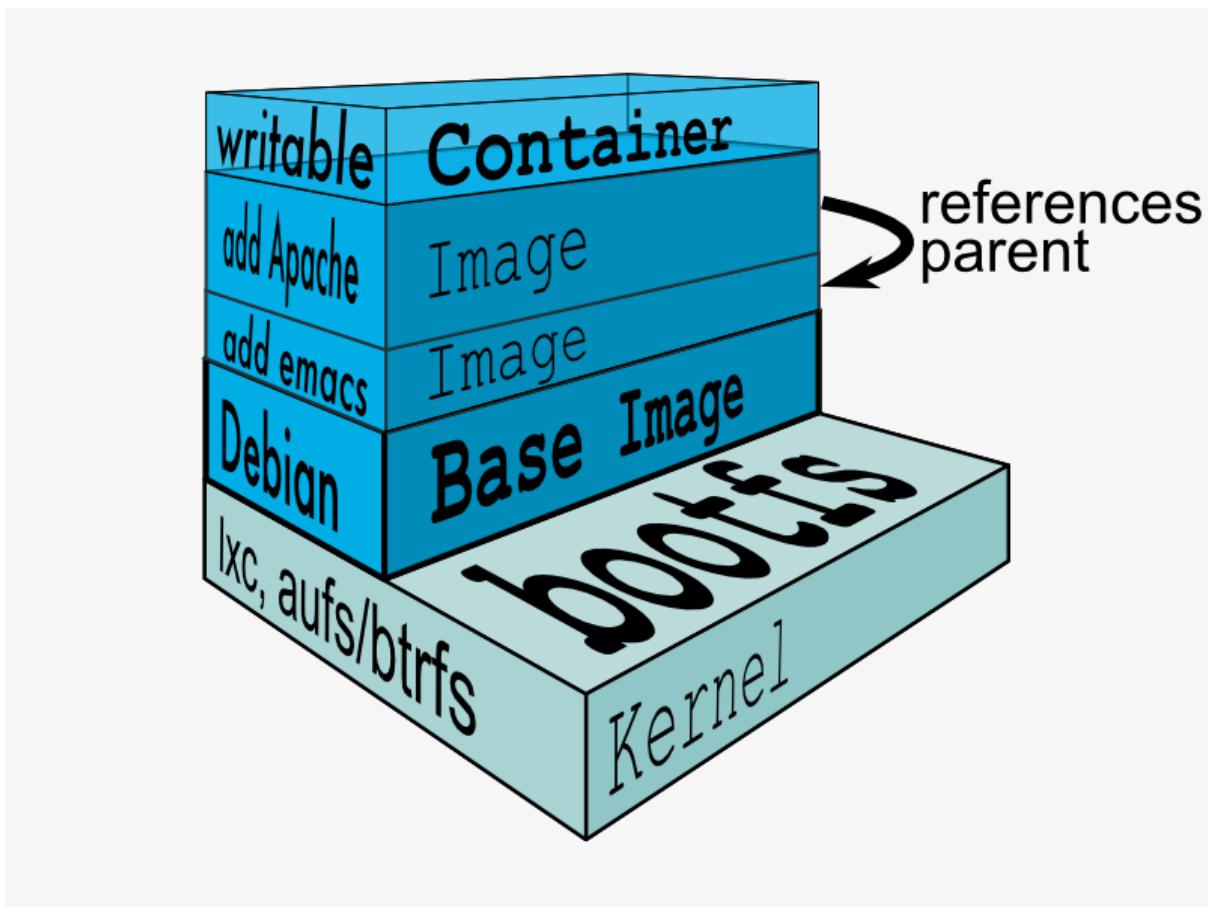
---

<sup>85</sup> Docker 0.9: Introducing execution drivers and libcontainer, su docker.com, 10 marzo 2014

<sup>86</sup> Chris Swan, Docker drops LXC as default execution environment, InfoQ, 13 marzo 2014

<sup>87</sup> Adron Hall, OSCON : Conversations, Deployments, Architecture, Docker and the Future?, CloudAve, 31 luglio 2013.

<sup>88</sup> Travis Reeder, How Docker Helped Us Achieve the (Near) Impossible, Iron.io, 22 aprile 2014



Di seguito, viene riportato un breve esempio tratto dalla documentazione ufficiale, in cui viene messa in piedi una piccola struttura per fornire un browser (Firefox) immediatamente disponibile su una connessione VNC.

```
# From: http://docs.docker.io/en/latest/use/builder/#dockerfile-examples
# Firefox over VNC
#
# VERSION 0.3

FROM ubuntu
# make sure the package repository is up to date
RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" >
/etc/apt/sources.list
RUN apt-get update

# Install vnc, xvfb in order to create a 'fake' display and firefox
RUN apt-get install -y x11vnc xvfb firefox
RUN mkdir /.vnc
```

```

# Setup a password
RUN x11vnc -storepasswd 1234 ~/.vnc/passwd
# Autostart firefox (might not be the best way, but it does the trick)
RUN bash -c 'echo "firefox" >> /.bashrc'

EXPOSE 5900
CMD ["x11vnc", "-forever", "-usepw", "-create"]

```

I Dockerfile contengono istruzioni per la creazione dei container, chiaramente, con un prefisso maiuscolo che indica il tipo di comando. Tra i fondamentali si trovano:

- **FROM:** Definisce da quale container partire per la creazione dell'immagine che andiamo a profilare;
- **RUN:** Esegue comandi (in genere Bash, ma in realtà di qualsiasi natura, in maniera dipendente dalla shell che abbiamo definito) all'interno della virtual machine;
- **EXPOSE:** Specifica le porte da esporre all'esterno per la reperibilità dei servizi in esecuzione all'interno del container.

In secondo luogo, i container creati sono stati trasferiti su DockerHub, piattaforma di hosting di container pubblici e privati da cui è possibile scaricare o caricare nuovi container in base alle esigenze di circostanza.

DockerHub è utilizzabile come un insieme di repository con le principali funzioni di gestione di versione.

Sulla macchina host su cui andrà in esecuzione il sistema, è stato semplicemente installato Docker e lanciati gli script per l'esecuzione dei servizi creati poco prima.

## 12. Sprint 7 - Valutazione

Il core del sistema è stato completato durante gli sprint precedenti, durante il meeting relativo all'ultimo sprint completato, sono state accettate dal team le ultime modifiche e spostate le issue nella pipeline “Done”. Il passo successivo è la valutazione delle prestazioni del sistema dal punto di vista dell'accuratezza nella trascrizione in un contesto reale e dal punto di vista delle prestazioni durante l'invio dei messaggi. Tale valutazione è stata quindi articolata in due fasi che vengono descritte in questo capitolo, la prima fase mira a verificare che l'accuratezza della trascrizione sia all'altezza di un contesto reale per il quale è stato studiato il sistema, la seconda mira a verificare che i tempi di invio e ricezione dei messaggi derivanti dalla trascrizione siano accettabili.

### 12.1. Fase 1: Accuratezza del riconoscimento

In questa fase è stata simulato un contesto reale, in particolare è stato chiesto a diversi relatori di esporre un discorso della durata media di un intervento in un seminario.

Per permettere la replicabilità del test, il discorso è stato scritto in modo che la durata media della sua lettura fosse maggiore di 5 minuti e minore di 10 ed è stato chiesto ai relatori sottoposti al test di leggerlo in maniera quanto più naturale possibile cercando di scandire le parole e le pause tra una parola e l'altra.

Per l'analisi dei risultati, è stato registrato il discorso in modo da poter contare le parole riconosciute correttamente e quelle riconosciute in modo scorretto. La registrazione serve anche a capire meglio in quali casi il riconoscimento non ha piena efficacia e se la causa del riconoscimento errato è dovuta al relatore od al sistema.

I soggetti presi in considerazione sono 10 e saranno denominati per facilità di riconoscimento attraverso la sigla S#. I soggetti considerati sono di sesso maschile e femminile in egual quantità.

Il microfono utilizzato è stato sempre il medesimo (smartphone OnePlus 2) per evitare differenze di riconoscimento dovute al dispositivo; le registrazioni effettuate sono state poi mandate in input diretto al sistema di riconoscimento tramite scheda audio esterna; infine è stato effettuato il conteggio delle parole correttamente trascritte identificate con la lettera C (Correct); le parole non riconosciute identificate con L (Lost); le parole riconosciute in maniera errata identificate con M (Misunderstood) e le parole riconosciute ma non pronunciate identificate con A (Added).

Al termine delle diverse sessioni di riconoscimento relative ciascuna ad una lettura differente, in maniera manuale sono state confrontate le parole pronunciate con quelle riconosciute e appuntato il conteggio.

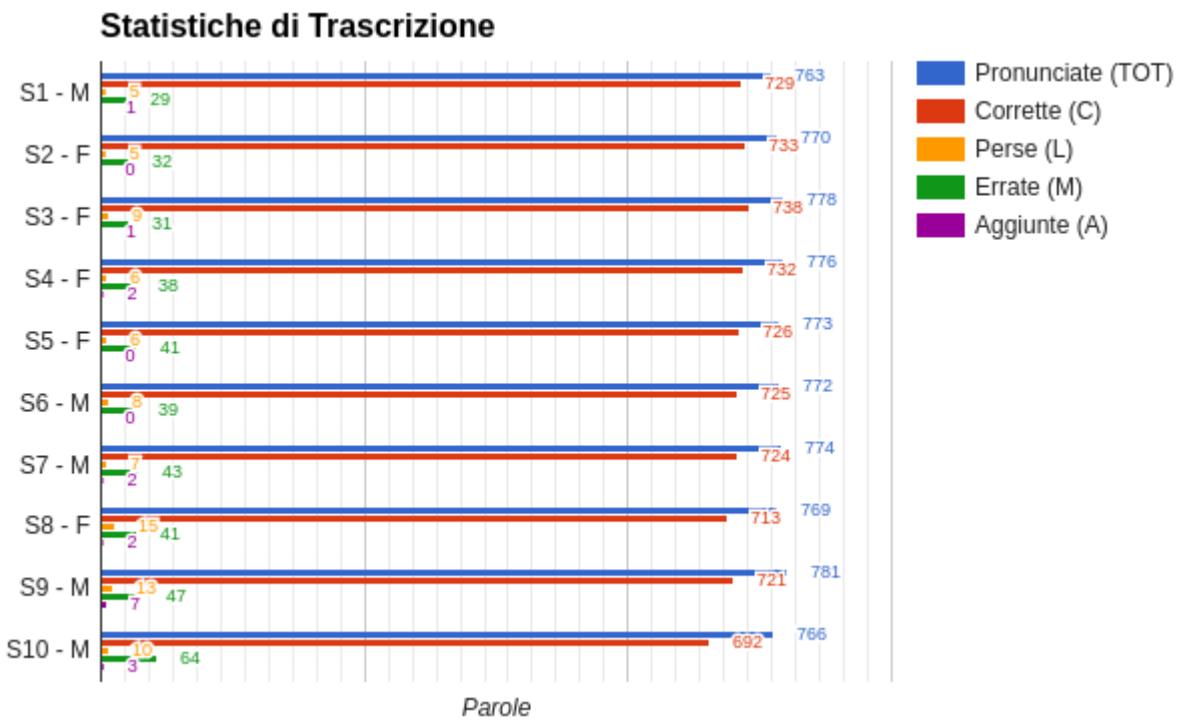
### Risultati della valutazione

Dopo i primi test, è emerso che il servizio di riconoscimento era abilitato a riconoscere esattamente 5 minuti di parlato per poi arrestarsi automaticamente, questo comportava evidentemente la perdita del resto del discorso. Per ovviare a questo problema, è stato aggiunto un timer che riavvia in maniera automatica il servizio di riconoscimento dopo 4:30 minuti. I test sono stati quindi ripetuti.

I risultati ottenuti (riportati in tabella sotto) sembrano essere molto confortanti poiché come si può osservare l'accuratezza calcolata come percentuale di parole correttamente riconosciute rispetto al totale delle parole riconosciute, si aggira intorno al 93%.

<b>Sogg. - Sesso</b>	<b>Pronunciate (TOT)</b>	<b>Corrette (C)</b>	<b>Perse (L)</b>	<b>Errate (M)</b>	<b>Aggiunte (A)</b>
<b>S1 - M</b>	763	729 (95,544 %)	5 (0,655 %)	29 (3,801 %)	1 (+0,131 %)
<b>S2 - F</b>	770	733 (95,195 %)	5 (0,649 %)	32 (4,156 %)	0 (+0,000 %)
<b>S3 - F</b>	778	738 (94,859 %)	9 (1,157 %)	31 (3,985 %)	1 (+0,129 %)
<b>S4 - F</b>	776	732 (94,330 %)	6 (0,773 %)	38 (4,897 %)	2 (+0,258 %)
<b>S5 - F</b>	773	726 (93,920 %)	6 (0,776 %)	41 (5,304 %)	0 (+0,000 %)
<b>S6 - M</b>	772	725 (93,912 %)	8 (1,036 %)	39 (5,052 %)	0 (+0,000 %)
<b>S7 - M</b>	774	724 (93,540 %)	7 (0,904 %)	43 (5,556 %)	2 (+0,258 %)
<b>S8 - F</b>	769	713 (92,718 %)	15 (1,951 %)	41 (5,332 %)	2 (+0,260 %)
<b>S9 - M</b>	781	721 (92,318 %)	13 (1,665 %)	47 (6,018 %)	7 (+0,896 %)
<b>S10 - M</b>	766	692 (90,339 %)	10 (1,305 %)	64 (8,355 %)	3 (+0,392 %)
<b>Media</b>	772,2	723,3	8,4	40,5	1,8
<b>Varianza</b>	29,7	168,9	11,6	100,5	4,4

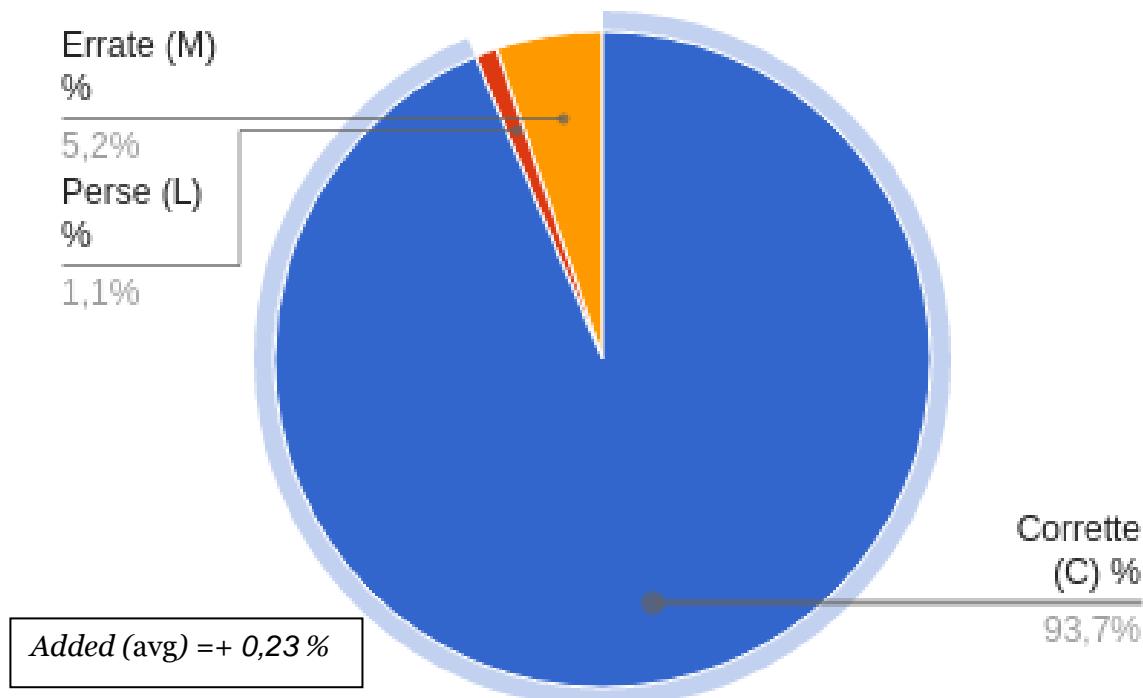
La tabella sopra riporta i risultati del conteggio delle parole riconosciute effettuato. Il primo valore riportato (TOT) rappresenta il numero di parole pronunciate da ciascuno speaker e comprende i valori C, L ed M (definiti precedentemente). Il valore A invece rappresenta le parole aggiunte dal sistema di riconoscimento derivanti probabilmente da rumore o voci estranee e non è conteggiato nel totale ma gestito a parte. Sotto, nel grafico, è possibile notare come effettivamente le parole correttamente riconosciute siano molto vicine a quelle pronunciate e come le parole riconosciute in maniera errata o non riconosciute siano quasi ininfluenti in percentuale.



Da una successiva analisi delle registrazioni e delle trascrizioni si evince come nella maggior parte dei casi, le parole denotate con L ovvero non riconosciute, siano dovute al riavvio del servizio di riconoscimento dopo i 4:30 minuti o dopo pause maggiori di 3 secondi oppure ancora a una pronuncia non corretta della parola. Spesso infatti si tende a legare durante la pronuncia di una frase le congiunzioni alla vocale iniziale o finale delle parole adiacenti o ancora a legare le vocali terminali delle stesse parole adiacenti; questo influisce inevitabilmente nel riconoscimento del genere della parola (maschile o femminile) o nella persona o coniugazione del verbo (es. vado/vada oppure comincia/cominciò). Da qui, inevitabilmente si evince il motivo delle parole riconosciute in maniera errata che solo in pochissimi casi si discostano semanticamente dall'originale, è invece errata nella maggior parte dei casi il prefisso o la desinenza o un suffisso.

Di seguito si riportano le percentuali di accuratezza ricavate e un grafico riassuntivo delle medie ottenute.

## Accuratezza di Trascrizione in Percentuale Media



Nel complesso, è possibile affermare che il sistema si adatta molto bene a diversi speaker di genere differente e con toni, timbri e intensità di voce differenti.

Il testo risultante dalla trascrizione è facilmente comprensibile anche con i piccoli errori di trascrizione rilevati. Il problema più importante da superare è sicuramente quello della punteggiatura, da non sottovalutare per la corretta comprensione di un testo. Per ovviare a questo problema è verosimile pensare ad un sistema di NLP (Natural Language Processing) accodato a quello di riconoscimento che elabori il testo sulla base di un dataset di frasi preimpostate nella lingua in cui si effettua il riconoscimento per migliorare la qualità sia della trascrizione, sia della comprensibilità del discorso trascritto.

### 12.2. Fase 2: Prestazioni del sistema

Per verificare le prestazioni del servizio di riconoscimento invece, sono stati calcolati i tempi trascorsi dall'invio del segnale audio al ritorno del risultato. In particolare, sono stati prima predisposti tre file audio estratti da una delle registrazioni effettuate per il test precedente, il file audio è stato quindi tagliato per durare 1 minuto e salvato a parte, poi è stato tagliato ancora per durare 30 secondi e salvato a parte, poi è stato tagliato un'ultima volta per durare 15 secondi e salvato in un terzo file. I file sono poi stati riprodotti mandando l'audio direttamente al sistema di riconoscimento e calcolando i seguenti tempi:

- **S-IR:** Start-InterimResult ovvero il tempo tra l'avvio del file audio e la ricezione del primo risultato provvisorio;
- **E-FR:** End-FinalResult ovvero il tempo trascorso tra il termine della traccia audio e la ricezione del risultato definitivo.

La prova è stata ripetuta per 10 volte per ognuno dei tre file audio e i risultati osservati sono riportati nella tabella sotto.

#### Risultati del test

	<b>Audio da 60 secondi</b>		<b>Audio da 30 secondi</b>		<b>Audio da 15 secondi</b>	
	<b>S-IR (sec.)</b>	<b>E-FR (sec.)</b>	<b>S-IR (sec.)</b>	<b>E-FR (sec.)</b>	<b>S-IR (sec.)</b>	<b>E-FR (sec.)</b>
<b>To1</b>	3	8	2	8	2	7
<b>To2</b>	2	8	2	8	3	7
<b>To3</b>	2	8	2	7	2	7
<b>To4</b>	2	7	2	8	2	7
<b>To5</b>	2	8	2	8	2	6
<b>To6</b>	2	8	3	8	2	7
<b>To7</b>	2	7	2	8	2	7
<b>To8</b>	2	8	2	7	2	6
<b>To9</b>	3	7	2	7	2	7
<b>T10</b>	2	8	2	8	2	6
<b>Media</b>	2,2	7,7	2,1	7,7	2,1	6,7
<b>Varianza</b>	0,2	0,2	0,1	0,2	0,1	0,2

Come è possibile notare dai dati ottenuti, il tempo di risposta con una trascrizione definitiva da parte del sistema di riconoscimento è di circa 8 secondi in generale, tale tempo scende se la durata dell'audio è molto breve (vedi Audio da 15 secondi).

Tuttavia, il tempo è dipendente dal riavvio del servizio di riconoscimento che viene effettuato se entro 3 secondi (valore impostato dallo sviluppatore) non si riceve alcun risultato provvisorio. Se tale timer non fosse impostato, il risultato finale giungerebbe ogni qualvolta il servizio avesse riconosciuto un periodo di silenzio.

## 13. Conclusioni e Sviluppi Futuri

L'obiettivo principale di questo lavoro di tesi era quello di riuscire a coinvolgere soggetti non udenti in contesti sociali tramite la trascrizione in tempo reale del parlato. Durante il lavoro di tesi è emerso che sono tanti i sistemi, sviluppati in tutto il mondo, a supporto dei non udenti, tuttavia tali sistemi sono molto costosi poiché forniscono il servizio mediante l'intervento di un soggetto umano (correttore) a monte del sistema di trascrizione automatica.

Si può affermare a questo punto che dal lavoro è emerso un sistema, "Scriba", capace di rendere semplice la fruizione di contenuti parlati tramite l'interfacciamento con un dispositivo mobile che tutti i giorni ognuno utilizza. Il sistema è in grado di fornire funzionalità di gestione della programmazione degli eventi tramite una comoda interfaccia web sia da PC Desktop, sia da un qualsiasi dispositivo mobile. Chi organizza eventi pubblici può quindi contare su Scriba per coinvolgere soggetti audiolesi in tali contesti senza dover necessariamente preoccuparsi di riservare i posti più vicini allo speaker per permettere a tali soggetti la lettura del labiale.

Il sistema nel complesso si comporta in maniera efficace e piuttosto efficiente (circa 93% di accuratezza), grazie all'utilizzo delle API messe a disposizione gratuitamente da Google, fornisce la trascrizione in tempo reale, in circa 7 secondi dal termine di una frase, di quello che il relatore dice, direttamente sugli smartphone dei partecipanti.

Ancora molto lunga è la strada da percorrere per ottenere un sistema che proponga una soluzione che sia possibile definire "sostitutiva" rispetto alle più costose soluzioni di respeaking. Il progetto è stato avviato come Open Source, per questo è sicuramente di buon auspicio la partecipazione di nuovi sviluppatori che collaborino al completamento dei task programmati a contorno della base fin ora sviluppata.

Tra le funzionalità che sicuramente si ritengono complementari e da implementare nell'immediato futuro si annoverano la gestione più "social" degli inviti che preveda quindi l'invito diretto (attraverso Scriba) di relatori e partecipanti ad un evento; la gestione di uno spazio temporale autonomo dedicato ad ogni relatore che potrà quindi gestire i propri interventi nei limiti stabiliti; la memorizzazione locale degli interventi passati da parte dei partecipanti che così saranno in grado di fruire del testo anche senza una connessione a internet; l'implementazione della seconda tipologia di eventi, quelli privati, che prevedranno l'accesso ai soli partecipanti invitati dal relatore e tante altre funzionalità di minore importanza.

Dal punto di vista tecnico, invece, è sicuramente possibile, attraverso uno studio approfondito e mirato, migliorare l'accuratezza della trascrizione attraverso un sistema

automatico di correzione che faccia uso di principi del Natural Language Processing (NLP) e che quindi contestualizzi il discorso per adattare correttamente la punteggiatura e la terminologia utilizzata.

Infine è auspicabile uno studio approfondito dell'esperienza fornita all'utente dal punto di vista della Human-Computer Interaction (HCI) che miri quindi a ristrutturare l'aspetto e la distribuzione delle funzionalità poste all'utente per fornire un sistema che sia altamente usabile in maniera quanto più piacevole possibile.