

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

CORSO DI LAUREA IN INFORMATICA



**Analisi, progettazione e sviluppo di un
motore conversazionale per una
piattaforma di gestione della forza lavoro**

Tesi di laurea triennale

Relatore

Prof. Claudio Enrico Palazzi

Laureando

Federico Perin

ANNO ACCADEMICO 2019-2020

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di 320 ore, dal laureando Federico Perin presso l'azienda AzzurroDigitale Srl. Lo scopo dello stage era quello di essere introdotto all'interno del progetto aziendale "Azzura.flow". Tale progetto prevede lo sviluppo di un bot denominato Azzurra, da integrare all'interno di una applicazione mobile. Azzurra quindi, attraverso una chat con l'utilizzatore umano svolgerà il ruolo di assistente offrendo funzionalità di supporto, come ad'esempio informare il lavoratore sul suo piano di lavoro.

Era richiesto come primo obbiettivo, acquisire le competenze tecniche richieste per poter contribuire allo sviluppo nel progetto attraverso lo studio e l'utilizzo di video lezioni offerte dalla piattaforma di e-learning Udeny.

In secondo luogo veniva richiesto lo studio del funzionamento dell'architettura del sistema che permette l'esecuzione di Azzurra, in particolare il funzionamento dei metodi del motore conversazionale denominato Azzura.io. Una volta apprese le conoscenze necessarie, si richiedeva la progettazione e l'implementazione di alcuni flussi di conversazione per Azzurra.

Affiancato alle attività di implementazioni era richiesto, da buona prassi, effettuare attività di documentazioni sia riguardante il codice ma anche di scelte progettuali, e lo sviluppo di una test-suite per l'applicazione mobile che ne verificasse il corretto funzionamento.

“If something’s important enough, you should try. Even if the probable outcome is failure.”

— Elon Musk

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Claudio Enrico Palazzi, relatore della mia tesi, per l’aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e nel avermi sempre sostenuto anche nei momenti difficili.

Infine desidero ringraziare i miei ex compagni di gruppo per il progetto didattico del corso di Ingegneria del Software, per aver reso meno pesanti le intere giornate passate a svolgere il progetto.

Padova, Settembre 2020

Federico Perin

Indice

| | | |
|----------|---|-----------|
| 1 | Introduzione | 1 |
| 1.1 | Convenzioni tipografiche | 1 |
| 1.2 | L'azienda AzzurroDigitale S.r.l | 1 |
| 1.3 | L'idea | 2 |
| 1.3.1 | Il contesto applicativo | 2 |
| 1.3.2 | Il progetto Azzurra.flow | 3 |
| 1.4 | Organizzazione del testo | 4 |
| 2 | Lo stage | 5 |
| 2.1 | Descrizione dello stage | 5 |
| 2.2 | Obiettivi | 6 |
| 2.2.1 | Classificazione | 6 |
| 2.2.2 | Definizione degli obiettivi | 6 |
| 2.3 | Prodotti attesi | 7 |
| 2.4 | Modalità di svolgimento del lavoro | 7 |
| 2.5 | Pianificazione del lavoro | 7 |
| 2.5.1 | Pianificazione settimanale | 7 |
| 2.6 | Variazioni | 9 |
| 2.7 | Strumenti e tecnologie utilizzate | 10 |
| 2.7.1 | Strumenti | 10 |
| 2.7.2 | Tecnologie | 11 |
| 2.8 | Motivazioni personali | 12 |
| 3 | Architettura del sistema AWMS | 13 |
| 3.1 | Descrizione | 13 |
| 3.1.1 | AWMS Dashboard | 14 |
| 3.1.2 | AWMS backend | 14 |
| 3.1.3 | Azzurra.io | 15 |
| 3.1.4 | Applicazione mobile | 16 |
| 3.2 | Operazioni | 19 |
| 3.2.1 | Creazione di una connessione attraverso WebSocket | 19 |
| 3.2.2 | Recupero di un flusso conversazionale | 20 |
| 3.2.3 | Richiesta e invio di dati | 21 |
| 3.2.4 | Gestione notifiche push | 22 |
| 4 | Azzurra Flow Engine | 25 |
| 4.1 | Scopo | 25 |
| 4.2 | Flussi di conversazione | 25 |

| | | |
|----------|--|-----------|
| 4.2.1 | Shortcuts “shortcuts” | 26 |
| 4.2.2 | Configurazione "config" | 27 |
| 4.2.3 | Blocchi per la conversazione | 27 |
| 4.2.4 | Oggetti ausiliari | 31 |
| 5 | Progettazione e codifica | 33 |
| 5.1 | Tecnologie e strumenti | 33 |
| 5.2 | Ciclo di vita del software | 33 |
| 5.3 | Progettazione | 33 |
| 5.4 | Design Pattern utilizzati | 33 |
| 5.5 | Codifica | 33 |
| 6 | Verifica e validazione | 35 |
| 7 | Conclusioni | 37 |
| 7.1 | Consuntivo finale | 37 |
| 7.2 | Raggiungimento degli obiettivi | 37 |
| 7.3 | Conoscenze acquisite | 37 |
| 7.4 | Valutazione personale | 37 |
| A | Appendice A | 39 |
| | Glossario | 41 |
| | Acronimi | 43 |
| | Bibliografia | 45 |

Elenco delle figure

| | | |
|------|---|----|
| 1.1 | Logo di AzzurroDigitale | 2 |
| 1.2 | Logo di AWMS | 2 |
| 1.3 | Logo del bot Azzurra | 3 |
| 3.1 | Architettura di sistema AWMS | 13 |
| 3.2 | Schermata di AWMS Dashboard | 14 |
| 3.3 | Sezione Questionario | 16 |
| 3.4 | Schede del questionario sulla salute | 17 |
| 3.5 | Schede dell'esito del questionario sulla salute | 17 |
| 3.6 | Sezione Profilo | 18 |
| 3.7 | Sezione Chat bot Azzurra | 18 |
| 3.8 | Sequence diagram per la creazione di una connessione attraverso WebSocket | 19 |
| 3.9 | Sequence diagram per il recupero di un flusso conversazionale | 20 |
| 3.10 | Sequence diagram per il recupero dei dati sul lavoratore | 21 |
| 3.11 | Sequence diagram per l'invio di notifiche push | 22 |
| 4.1 | Menu contenente le shortcuts disponibili | 26 |

Elenco delle tabelle

| | | |
|-----|--|---|
| 2.1 | Tabella del tracciamento dei requisiti qualitativi | 9 |
|-----|--|---|

Capitolo 1

Introduzione

1.1 Convenzioni tipografiche

Nella stesura del presente documento, sono state adottate le seguenti convenzioni tipografiche:

- * gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- * per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*^[g];
- * i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

1.2 L'azienda AzzurroDigitale S.r.l

Lo stage è stato svolto nell'azienda AzzurroDigitale S.r.l. situata nella zona industriale di Padova. AzzurroDigitale nasce nel 2015 quando tre giovani padovani (Carlo Pasqualetto, Jacopo Pertile e Antonio Fornari) fondano la startup, puntando fortemente nelle nuove emergenti tecnologie che il mercato offriva. Come primo cliente, fu l'azienda Electrolux che grazie a una forte attività collaborazione, fu sviluppata una piattaforma per la gestione degli operai denominata AWMS, che tutt'ora continua a ricevere miglioramenti e a crescere. Dopo il successo ottenuto con la collaborazione con Electrolux, l'azienda capisce che il mercato delle aziende manifatturiere è la nicchia sulla quale puntare soprattutto grazie al momento storico della digital transformation.



Figura 1.1: Logo di AzzurroDigitale

Oggi AzzurroDigitale offre servizi di industrial digital transformation, workforce management e people empowerment, con l'obiettivo comune di aiutare le aziende manifatturiere a migliorare e implementare i loro processi grazie alle tecnologie, non intese come sostitutive all'uomo, ma bensì come mezzi che abilitano le persone a lavorare nel miglior modo possibile, massimizzando lo sforzo lavorativo.

1.3 L'idea

1.3.1 Il contesto applicativo

L'azienda AzzurroDigitale offre come principale servizio, la piattaforma di gestione forza lavoro denominata AWMS.

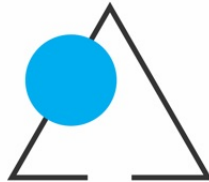


Figura 1.2: Logo di AWMS

AWMS è una soluzione software che utilizza algoritmi di machine learning, per risolvere uno dei problemi cardine di un plant manager ovvero, la pianificazione ottimale della forza lavoro che ha disposizione. L'obiettivo principale della soluzione è quello di pianificare la persona giusta al posto giusto in base alle competenze tecniche possedute del lavoratore. Per permettere il funzionamento della pianificazione, la piattaforma estrae da database interni dell'azienda, dati sui lavoratori che ne descrivono le competenze che possiedono. Viene perciò registrato uno storico per ogni lavoratore che se nel tempo acquisirà nuove competenze queste verranno indicate nei dati dei database, aggiornandoli. In base perciò, ai dati estratti dalla piattaforma viene scelto il

miglior candidato per un determinato compito. AWMS offre la possibilità di pianificare il lavoro per il giorno successivo ma anche gestire situazioni impreviste, come ad esempio l'assenza di un lavoratore.

1.3.2 Il progetto Azzurra.flow

Il progetto Azzurra.flow nasce dalla esigenza, da parte dell'azienda AzzurroDigitale, di offrire un prodotto completo per tutti i soggetti coinvolti nelle attività lavorative. Con AWMS si ha uno strumento che supporta i team leader o i plant manager nella loro pianificazione del lavoro ma non si ha nessun strumento che supporti il lavoratore. Da questa mancanza nasce perciò il progetto "Azzurra.flow". Esso consiste nel creare un bot denominato Azzurra, inserito in un'applicazione mobile, che permette di offrire delle funzionalità utili all'utente che sono:

- * Visualizzare il proprio turno di lavoro;
- * Visualizzare i propri permessi lavorativi o richiederne di nuovi;
- * Visualizzare avvisi da parte dell'azienda;
- * Sapere informazione sul menu del giorno della mensa aziendale;
- * Poter effettuare prenotazioni di un posto in una sala riunioni e visualizzare le proprie prenotazioni, inoltre utilizzare un scannerizzatore QR-CODE per riscattare il posto prenotato.

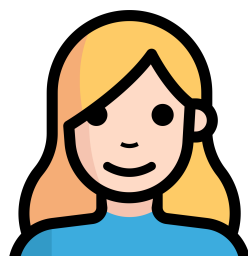


Figura 1.3: Logo del bot Azzurra

Il progetto include non solo lo sviluppo dell'applicazione mobile con Azzurra ma, un motore conversazionale in grado di poter generare una conversazione con il lavoratore, attraverso dei flussi di conversazione, anche essi da sviluppare, che indicano che cosa deve fare Azzurra, dopo essere stati interpretati dal motore conversazionale. Inoltre questi flussi devono essere memorizzati in un preciso posto perciò, è stato progettato che sia un database contenuto nella nuova componente Azzurra.io, la quale ha il compito di non solo di tenere memorizzati i flussi di conversazioni esistenti e di inviarli a Azzurra quando li richiede, per sapere che messaggi generare, ma di fare da tramite tra l'applicazione con all'interno Azzurra e AWMS tutto attraverso una comunicazione tramite WebSocket.

1.4 Organizzazione del testo

Il capitolo trattato attualmente è l'introduzione del documento, dove si è spiegato brevemente l'ambito di lavoro e il progetto sul quale si è svolto lo stage.

Di seguito il documento sarà organizzato nella seguente struttura:

Il secondo capitolo descrive in modo dettagliato lo stage svolto, indicandone obiettivi, pianificazione, strumenti e tecnologie utilizzate. Infine verranno esposte le motivazioni per cui ho scelto di svolgere questo stage.

Il terzo capitolo descrive l'architettura del sistema AWMS che permette il funzionamento di Azzurra.

Il quarto capitolo approfondisce il funzionamento del motore conversazionale di Azzurra, indicando perciò come avviene una conversazione.

Il quinto capitolo descrive il lavoro di analisi, progettazione e implementazione dei flussi di conversazione per Azzurra.

Il sesto capitolo descrive le tecnologie utilizzate per costruire una test-suite per Azzurra, espone il piano di test stabilito inserendo i risultati ottenuti.

Il settimo capitolo rappresenta la conclusione del documento, viene perciò riepilogato il lavoro svolto durante lo stage, gli obiettivi raggiunti e infine una valutazione personale sull'esperienza di stage.

Capitolo 2

Lo stage

Nel seguente capitolo verrà descritto in dettaglio la proposta di stage accetta, indicandone gli obiettivi, la pianificazione, i prodotti attesi e gli strumenti e tecnologie utilizzate durante lo stage, e infine verranno esposte le motivazioni per cui ho scelto questo stage.

2.1 Descrizione dello stage

Lo stage era legato al progetto interno dell'azienda denominato "Azzurra.flow". Tale progetto nasceva dall'esigenza dell'azienda AzzurroDigitale di offrire un prodotto più completo ai propri clienti da affiancare alla soluzione AWMS. Perciò era previsto di implementare un'applicazione mobile che potesse comunicare con la piattaforma AWMS, dando un mezzo di supporto al lavoratore di una azienda manifatturiera. All'interno di essa doveva essere implementato una chat bot con un bot denominato "Azzurra" che offra funzionalità di supporto al lavoratore. All'interno del progetto era anche previsto le implementazioni necessarie per la comunicazione tra AWMS e l'applicazione mobile, quindi la gestione di una connessione websocket e la creazione della componente Azzurra.io, la quale ha il compito di tenere memorizzati i flussi conversazionali esistenti e di inviarli a Azzurra quando li richiede, per sapere che messaggi devono essere generati, e di fare da tramite tra l'applicazione mobile e AWMS.

Partendo dal progetto "Azzurra.flow" è stato creato lo stage da me sostenuto, composto da attività che andassero a contribuire allo sviluppo del progetto.

Lo stage è stato costruito inserendo le seguenti parti:

- * La prima parte era stato pianificato lo studio delle tecnologie che sarebbero state utilizzate durante lo stage e nella contribuzione dello sviluppo del progetto "Azzurra.flow". Lo studio autonomo delle tecnologie era supportato da video lezioni della piattaforma di e-learning Udeny, offerte dall'azienda;
- * La seconda parte era dedicata allo studio del funzionamento dell'architettura del sistema che permette l'esecuzione di Azzurra, in particolare il funzionamento dei metodi del motore conversazionale denominato Azzurra.io e in più come esercitazione, era richiesto la creazione di alcuni test e2e per la parte frontend del sistema quindi l'implementare di test per la dashboard di AWMS;
- * La terza parte era dedicata all'analisi, progettazione e implementazione dei flussi di conversazione per il bot Azzurra;

- * La quarta parte era dedicata alla stesura della documentazione per la Solution Design di Azzurra;
- * La quinta parte sulle basi che si era in parto nella seconda parte, era previsto lo sviluppo di una test-suite di test e2e, con l'obbiettivo di testare in modo automatizzato, se le funzionalità dell'applicazione mobile funzionassero in modo corretto;
- * Infine nella sesta parte era dedicata allo studio di alcuni aspetti dell'applicazione mobile che sono:
 - La gestione delle notifiche push;
 - Il template engine multi-lingua;
 - La gestione comportamenti mobile app in condizioni di mancanza di connettività.

2.2 Obiettivi

2.2.1 Classificazione

Piano di lavoro per il progetto di stage dell'anno accademico 2019/2020 svolto presso AzzurroDigitale, dove si farà riferimento ai requisiti secondo le seguenti notazioni:

- * *OB-x* per i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
- * *OD-x* per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- * *OF-x* per i requisiti facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Dove x è un numero progressivo intero maggiore di zero.

2.2.2 Definizione degli obiettivi

Si prevede lo svolgimento dei seguenti obiettivi:

Obbligatori

- * **OB-1:** competenza nello sviluppo delle singole attività identificate con i linguaggi PHP e Typescript.

Desiderabili

- * **OD-1:** capacità autonoma di analisi delle singole attività delle soluzioni tecniche viste durante il progetto;
- * **OD-2:** capacità autonoma di progettazione delle singole attività delle soluzioni tecniche viste durante il progetto.

2.3 Prodotti attesi

Durante lo stage era atteso lo sviluppo dei seguenti deliverable:

- * **Analisi tecnica:** Descrizione dell'analisi svolta e soluzione identificata, sarà redatta sulla piattaforma documentale aziendale Confluence;
- * **Software:** Implementazione software della soluzione identificata, redatta con l'IDE di sviluppo identificato per il progetto e depositata sul repository GitLab di riferimento.

2.4 Modalità di svolgimento del lavoro

Lo stage è stato svolto in presenza negli uffici di AzzurroDigitale rispettando tutte le norme sul distanziamento sociale. L'orario di lavoro è stato dalle 9:00 fino alle 13:00 e dalle 14:00 fino alle 18:00. Durante lo stage sono stato inserito in un gruppo di sviluppatori, i quali fornivano una azione di supporto e guida nel caso in cui sorgevano difficoltà nel proseguimento delle attività di stage. Nonostante ciò ero comunque seguito anche dal mio tutor aziendale non che team leader del gruppo di sviluppatori, il quale esplicitava i task che dovevo realizzare e gli obiettivi attesi nello svolgimento di ogni task.

Durante lo stage per gestire le attività di progetto è stato utilizzato il modello agile SCRUM, modello adottato dall'azienda per gestire i propri progetti. Vi erano quindi le seguenti attività:

- * Daily meeting mattutino, della durata di circa 15 minuti, dove vengono discussi i task della giornata, ed eventuali problemi bloccanti;
- * Weekly review dove vengono analizzate e discusse le attività che dovevo svolgere nella settimana successiva.

Infine, durante lo stage era mio compito redigere un registro su cui, quotidianamente, segnare le attività svolte.

2.5 Pianificazione del lavoro

Di seguito viene mostrato in dettaglio la pianificazione delle attività per i mesi di Luglio, Agosto e Settembre 2020. Per ognuna delle seguenti attività si dovrà:

- * Leggere e comprendere l'analisi funzionale;
- * Analizzare, progettare e documentare la soluzione tecnica identificata;
- * Contribuire all'implementazione della soluzione tecnica;
- * Contribuire all'implementazione ed all'esecuzione test e bugfix.

2.5.1 Pianificazione settimanale

Di seguito viene riportata la pianificazione completa, basata su 320 ore, delle attività svolte durante lo stage:

Prima Settimana 01/07-03/07 (24 ore)

- * **Formazione Angular:** corso Udemy + review di alcuni componenti di AWMS;
- * **Formazione Ionic:** corso Udemy + review di alcuni componenti di AWMS Azzurra (mobile app).

Seconda Settimana 06/07-10/07 (40 ore)

- * **Formazione NestJS:** corso Udemy + review di alcuni componenti di “Azzurra” già sviluppati;
- * **Unit testing:** (Jasmine+Karma) lato frontend;
- * **End-to-end testing:** (Appium+Cucumber.js) lato mobile app.

Terza Settimana 13/07-17/07 (40 ore)

- * Approfondimenti architetture a micro-services e loro implementazione in AWMS Platform;
- * Analisi implementazione di un conversational flow editor visuale;
- * Software selection (con test/poc) per lo sviluppo di un conversational flow editor visuale.

Quarta Settimana 20/07-24/07 (40 ore)

- * Contributi alla redazione della Solution Design di “Azzurra”;
- * Contributi alla documentazione sorgenti di “Azzurra” (frontend/backend).

Quinta Settimana 27/07-31/07 (40 ore)

- * Review di alcuni componenti di AWMS;
- * Aspetti di scalabilità di un flow-engine (concorrenzialità, HA, persistenza/storizzazione messaggi)

Sesta Settimana 03/08-07/08 (40 ore)

- * Contributi alla redazione della Solution Design di “Azzurra”;
- * Implementazione Push Notifications (lato mobile App);
- * Implementazione Push Notifications (lato backend).

Settima Settimana 17/08-21/08 (40 ore)

- * Progettazione e documentazione template engine multi-lingua;
- * Implementazione template engine multi-lingua (l’assistente virtuale dovrà avere il supporto multi-lingua) basato su sintassi “mustache”.

Ottava Settimana 24/08-28/08 (40 ore)

- * Gestione comportamenti mobile app in condizioni di mancanza di connettività (corner cases, messaggi di feedback, landing pages).

Nona Settimana 31/08-01/09 (16 ore)

- * Continuazione ottava settimana.

Di seguito viene riportata una tabella riassuntiva della pianificazione:

| Durata in ore | Data inizio - fine | Attività |
|---------------|-------------------------|---|
| 24 | 01/07/2020 - 03/07/2020 | Studio delle tecnologie, Angular 2+ e Ionic, da utilizzare durante lo stage. |
| 40 | 06/07/2020 - 10/07/2020 | Studio di componenti del dell'architettura di sistema di Azzurra, creazione di test per la dashboard di AWMS e per l'applicazione mobile. |
| 40 | 13/07/2020 - 17/07/2020 | Continuazione studio delle componenti del sistema di Azzurra e analisi, progettazione e implementazione di flussi conversazionali. |
| 40 | 20/07/2020 - 24/07/2020 | Scritture di documentazione per le componenti di Azzurra. |
| 40 | 27/07/2020 - 31/07/2020 | Continuazione di altre componenti di AWMS. |
| 40 | 03/08/2020 - 07/08/2020 | Documentazione delle componenti AWMS e implementazione notifiche push. |
| 40 | 17/08/2020 - 21/08/2020 | Progettazione, implementazione e documentazione di template engine multi-lingua. |
| 40 | 24/08/2020 - 28/08/2020 | Gestione comportamenti mobile app in condizioni di mancanza di connettività. |
| 16 | 31/08/2020 - 01/09/2020 | Continuazione ottava settimana. |

Tabella 2.1: Tabella del tracciamento dei requisiti qualitativi

2.6 Variazioni

Nella seconda settimana è stato deciso di svolgere al posto di test d'unità nella parte front-end, test end to end con lo scopo di esercitazione. La creazione di test end to end per l'applicazione mobile è stata spostata alla quinta settimana per poter testare anche i flussi di conversazione implementati per la chat con Azzurra.

2.7 Strumenti e tecnologie utilizzate

2.7.1 Strumenti

HTML

L'HTML è un linguaggio di markup per la strutturazione delle pagine web. Nato per la formattazione e impaginazione di documenti ipertestuali disponibili nel web 1.0, oggi è utilizzato principalmente per il disaccoppiamento della struttura logica di una pagina web. Attualmente HTML5 è l'ultima versione di HTML la quale porta una sintassi più semplice e un pieno supporto anche a browser più datati.

CSS

È un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML ad esempio i siti web e relative pagine web. Permette una programmazione più chiara e facile da utilizzare, sia per gli autori delle pagine stesse sia per gli utenti, garantendo anche il riutilizzo di codice e facilita la manutenzione. Le specifiche CSS3 sono costituite da sezioni separate dette "moduli" e hanno differenti stati di avanzamento e stabilità.

TypeScript

È un linguaggio di programmazione open source che estende la sintassi di JavaScript in modo che qualunque programma scritto in JavaScript sia anche in grado di funzionare con TypeScript senza nessuna modifica. Come JavaScript è un linguaggio di programmazione orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso.

Angular 2+

Angular è un framework open source per lo sviluppo di applicazioni web con licenza MIT, sviluppato principalmente da Google. Angular è l'evoluzione di AngularJS infatti, è stato completamente riscritto rispetto a AngularJS e le due versioni non sono compatibili. Il linguaggio di programmazione usato per AngularJS è JavaScript mentre quello di Angular è TypeScript. Angular è stato progettato per fornire uno strumento facile e veloce per sviluppare applicazioni che girano su qualunque piattaforma inclusi smartphone e tablet. Inoltre le applicazioni sviluppate in Angular vengono eseguite interamente dal web browser dopo essere state scaricate dal web server. Questo comporta il risparmio di dover spedire indietro la pagina web al web-server ogni volta che c'è una richiesta di azione da parte dell'utente.

Ionic

Ionic è un SDK open source completo per lo sviluppo di app mobili ibride e permette di essere utilizzato con qualsiasi framework per lo sviluppo di applicazioni web. Ionic fornisce strumenti e servizi per lo sviluppo di applicazioni web ibride mobili, desktop e progressive basate su moderne tecnologie e pratiche di sviluppo web, utilizzando tecnologie web come CSS, HTML5 e Sass. In particolare, le app mobili possono essere

costruite con queste tecnologie Web e quindi distribuite tramite app store nativi per essere installate sui dispositivi mobili, utilizzando Cordova o Capacitor.

Protractor

Protractor è un framework di test end-to-end per applicazioni Angular e AngularJS. Protractor esegue test sulla applicazione in esecuzione in un browser reale, interagendo con essa come farebbe un utente.

Appium

Appium è un strumento open source che permette di eseguire in modo automatizzato script per testare applicazioni native, applicazioni web mobile e applicazioni ibride su Android o iOS utilizzando un webdriver.

Cucumber

Cucumber è un strumento che permette di creare test automatizzati con una specifica non ambigua e documenta come si comporta effettivamente il sistema. Cucumber supporta lo sviluppo guidato dal comportamento (BDD).

Selenium

Selenium è un framework che permette di testare le applicazioni web. Selenium fornisce uno strumento di riproduzione per la creazione di test funzionali senza la necessità di apprendere un linguaggio di scripting di test (Selenium IDE). Fornisce anche un linguaggio specifico del dominio di test (Selenese) per scrivere test in altri linguaggi di programmazione, come C# , Groovy , Java , Perl , PHP , Python , Ruby e Scala. I test possono quindi essere eseguiti sulla maggior parte dei browser Web moderni.

Npm

Npm è un gestore di pacchetti per il linguaggio di programmazione JavaScript. È il gestore di pacchetti predefinito per l'ambiente di runtime JavaScript Node.js. Consiste in un client da linea di comando, chiamato anch'esso npm, e un database online di pacchetti pubblici e privati.

2.7.2 Tecnologie

WebStorm

WebStorm è un ambiente di sviluppo integrato progettato per lo sviluppo web, principalmente in JavaScript e TypeScript. Supporta comunque, altri linguaggi per lo sviluppo di applicazioni web come ad esempio HTML, CSS, e PHP.

Jira Software

È un software proprietario che consente il bug tracking e la gestione dei progetti agile sviluppato da Atlassian.

Jira Confluence

È una piattaforma collaborativa sviluppata da Atlassian e scritta in Java, dove vengono forniti i strumenti per la scrittura e gestione della documentazione.

GitLab

È una piattaforma web open source che permette la gestione di repository Git e di funzioni trouble ticket.

2.8 Motivazioni personali

Attraverso la partecipazione all'iniziativa di StageIT, organizzata dall'Università di Padova e da Assindustria venetocentro, ho potuto entrare in contatto con molte aziende del territorio. Durante la partecipazione telematica all'evento ero alla ricerca di un'azienda che proponesse un progetto di stage con le seguenti caratteristiche:

- * permettermi di ampliare e migliorare le mie conoscenze in Angular ma più in generale a imparare a utilizzare nuove tecnologie per lo sviluppo front-end;
- * che trattasse tematiche legate allo sviluppo di applicazioni mobile;
- * permettermi di lavorare in un ambiente giovane e dinamico.

Confrontando con le varie aziende con cui ero entrato in contatto ho scelto di accettare lo stage proposto da AzzurroDigitale.

Questo perché nella loro proposta di stage c'è tutti i tre punti elencati prima, infatti grazie a questo progetto di stage ho avuto modo di migliorarmi nell'utilizzo di Angular imparando a utilizzare i metodi offerti da lui, in modo più efficiente. Inoltre, ho avuto la possibilità di sviluppare un'applicazione mobile grazie all'utilizzo di Ionic e Cordova. Altro aspetto importante fu che quest'azienda si distingue dal fatto che per gestire i propri progetti utilizza la metodologia agile SCRUM, una tematica mi interessava scoprire come valida alternativa al modello incrementale appreso durante il progetto del corso di Ingegneria del Software. Infine, l'azienda è una realtà giovane nata da meno di 5 anni fatta da persone giovani in cui potevo inserirmi facilmente.

Capitolo 3

Architettura del sistema AWMS

In questo capitolo verranno descritti tutte le componenti dell'architettura AWMS e le varie operazioni di comunicazione tra le varie componenti.

3.1 Descrizione

Come scritto precedentemente, dietro all'applicazione mobile c'è tutta un architettura di sistema che permette la comunicazione tra la piattaforma AWMS e l'applicazione mobile con Azzurra.

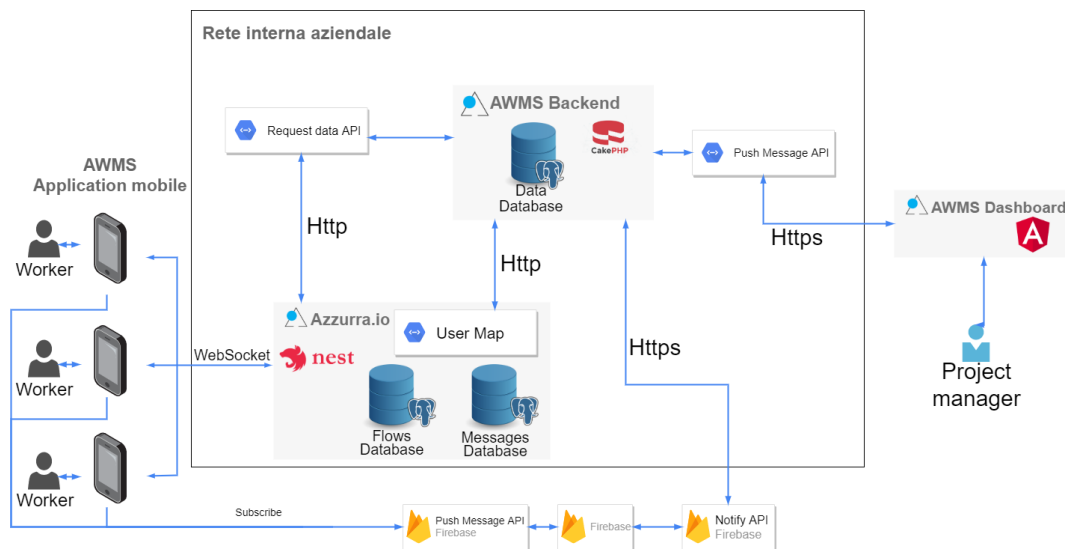
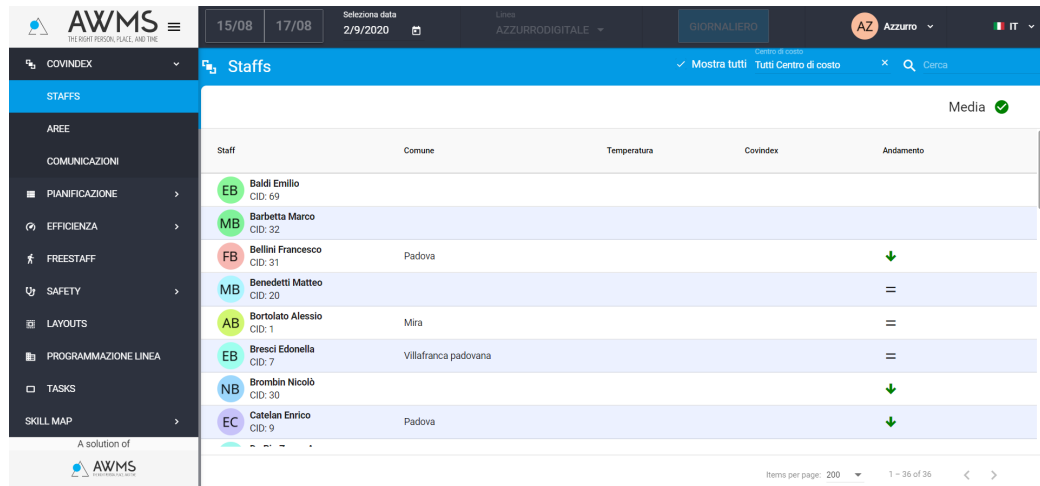


Figura 3.1: Architettura di sistema AWMS

La figura precedente illustra come è composta l'architettura, dove ogni componente verrà descritta nelle successive sotto sezioni.

3.1.1 AWMS Dashboard

È il pannello di controllo attraverso il quale un project manager può interagire con la piattaforma AWMS per poter pianificare il lavoro da svolgere, cioè assegnare un compito alla persona più idonea. Il pannello di controllo è una applicazione web che è stata sviluppata in Angular. La dashboard per comunicare con il back-end,



The screenshot shows the AWMS Dashboard interface. On the left is a sidebar menu with options like COVINDEX, STAFFS, AREE, COMUNICAZIONI, PIANIFICAZIONE, EFFICIENZA, FREESTAFF, SAFETY, LAYOUTS, PROGRAMMAZIONE LINEA, TASKS, and SKILL MAP. The main area displays a table of staff members. The table has columns for Staff, Comune, Temperatura, Covindex, and Andamento. The data is as follows:

| Staff | Comune | Temperatura | Covindex | Andamento |
|---------------------------------|----------------------|-------------|----------|-----------|
| EB Baldi Emilio CID: 69 | | | | |
| MB Barbetta Marco CID: 32 | | | | |
| FB Bellini Francesco CID: 31 | Padova | | | ↓ |
| MB Benedetti Matteo CID: 20 | | | | = |
| AB Bortolato Alessio CID: 1 | Mira | | | = |
| EB Bresci Edonella CID: 7 | Villafranca padovana | | | = |
| NB Brombin Nicolò CID: 30 | | | | ↓ |
| EC Catelan Enrico CID: 9 | Padova | | | ↓ |

At the bottom right of the table, it says 'Items per page: 200' and '1 - 36 of 36'.

Figura 3.2: Schermata di AWMS Dashboard

utilizza delle Application Program Interface (API)⁸ che il back-end espone, quindi per una ragione di sicurezza, back-end e l'applicazione web cioè il front-end, comunicano attraverso API² con in più l'utilizzo del protocollo di comunicazione HTTPS che cripta la comunicazione. Nella Figura 3.1 viene mostrato il caso in cui il front-end utilizza API³ per l'invio di notifiche push, questo perché è previsto che una volta il team leader sceglie il lavoratore più idoneo per un certo lavoro, il lavoratore deve essere avvisato, così sarà compito del front-end avvisare il back-end che c'è stata una nuova assegnazione e che questa assegnazione deve essere comunicata al diretto interessato attraverso un notifica sull'applicazione mobile con all'interno Azzurra.

3.1.2 AWMS backend

Come dice il suo nome, AWMS backend rappresenta il backend del sistema. AWMS backend è sviluppato usando lo strumento CakePHP, un framework per lo sviluppo di applicazioni web scritto in PHP. Al suo interno risiede il database che contiene tutte le informazioni sui lavoratori e tra questi quindi, i dati da mostrare nei messaggi di Azzurra come ad esempio il piano di lavoro che ha il lavoratore in uno specifico giorno, qualora ne venga fatta richiesta. Il database utilizza come DBMS PostgreSQL.

Come detto al punto precedente, per comunicare con il backend, esso espone delle API⁴ per la comunicazione infatti, esiste un API⁵ per l'invio di notifiche push ma esiste anche un API⁶, utilizzata da Azzurra.io, per la richiesta di informazioni sul lavoratore necessarie per completare il flusso di conversazione. Quindi questa API⁷ permetterà di richiedere dati al backend che li andrà a cercare nel suo database interno che se l'interrogazione al database da esito positivo, ritornerà le informazioni richieste a Azzurra.io. Il backend si trova all'interno della rete interna dell'azienda che ha

acquistato la soluzione di AzzurroDigitale, anche Azzurra.io è all'interno della rete, perciò tra queste due componenti avviene attraverso il protocollo di comunicazione HTTP. Il backend ha la possibilità di comunicare direttamente con Azzurra.io quando deve inviare una notifica push e ha necessità di sapere quali utenti sono attivi, cioè hanno una connessione aperta con Azzurra.io. Per gli utenti invece che non hanno una connessione aperta con Azzurra.io e quindi non sono attivi l'invio della notifica verrà fatto utilizzando i servizi offerti da Firebase, la cui comunicazione tra backend e Firebase avviene tramite HTTPS perché Firebase è un servizio esterno. La gestione l'invio delle notifiche push verrà comunque tratta in modo più dettagliato più avanti nel seguente capitolo.

3.1.3 Azzurra.io

Azzurra.io è una componente strategica per il funzionamento del bot Azzurra. Azzurra.io è sviluppata attraverso il framework NestJS. Al suo interno ha due database con DBMS PostgreSQL. Il primo database contiene i flussi di conversazione i quali indicano al bot che sequenza di passi deve fare durante la conversazione con l'utente umano, la loro struttura verrà spiegata in modo dettagliato al capitolo successivo. Il secondo database permette di memorizzare i messaggi fatti tra il bot Azzurra e l'utente umano. La scelta di adottare quest'ultimo database è dettata dalle seguenti motivazioni:

- * Per mantenere lo stato della conversazione cioè, se l'utente decide di non andare avanti con la conversazione e di continuarla in un secondo momento, grazie a questo database, in cui viene salvato lo stato della conversazione, potrà continuare la conversazione da dove l'aveva lasciata. Il mantenimento dello stato della conversazione non ha tempo illimitato ma dura al massimo un ora dopo di che lo stato verrà cancellato;
- * Per una migliore user experience si è scelto, nel caso in cui ci siano state delle conversazioni in precedenza, di mostrare i messaggi delle conversazioni precedenti, così che se l'utente ha bisogno di un'informazione che ha già chiesto precedentemente ma che si è dimenticato, basta che controlli i messaggi presenti nella chat senza dover richiedere ad Azzurra l'informazione dimenticata.

La connessione tra l'applicazione mobile e Azzurra.io è possibile grazie ai WebSocket che permettono di aprire una connessione tra i due e di mantenere sempre aggiornati i dati ad esempio la struttura dei flussi di conversazione, qualora venissero aggiornati. Per tenere traccia dei utenti connessi con Azzurra.io tramite l'applicazione mobile, viene utilizzata la mappa chiave-valore, interna ad Azzurra.io denominata User Map, la quale servirà a rispondere alle richieste del backend quando avrà bisogno della lista di utenti attivi per l'invio della notifica push. Come detto all'inizio del punto Azzurra.io è un componente strategico per principalmente due motivi.

- * Quando si vuole aggiungere un nuovo flusso conversazionale o modificare un flusso già esistente, se non esistesse Azzurra.io, questi sarebbero salvati nell'applicazione che ne comporterebbe l'aggiornamento dell'applicazione mobile e quindi effettuare una nuova pubblicazione nell'Play Store per i dispositivi Android e nel Apple Store per i dispositivi iOS ad ogni aggiunta o modifica dei flussi. Grazie all'esistenza di Azzurra.io ciò viene evitato perché esiste il database dedicato per la memorizzazione dei flussi conversazionali che se c'è da aggiungere un nuovo flusso basta semplicemente inserirlo all'interno del database, analogamente per

la modifica di un flusso. Inoltre, grazie alla connessione tramite WebSocket qualunque modifica o aggiunta viene subito recepita dall'applicazione mobile;

- * Per evitare che vengano fatte un numero elevato di richieste al backend si è deciso di distribuire le informazioni in diverse componenti della rete, infatti il bot per sapere che flusso conversazionale deve seguire per generare i messaggi per la conversazione con l'utente umano, chiede a Azzurra.io e non al backend. Il backend però verrà contattato quando il bot Azzurra ha bisogno di dati sul lavoratore da mostrare, questa richiesta però sarà fatta inizialmente a Azzurra.io che si prenderà carico di richiedere le informazioni al backend e di ritornarle all'applicazione. Quindi il backend sarà contattato solo dalla dashboard e da Azzurra.io per il caso descritto precedentemente per il processo di autenticazione dell'utente, diminuendo il carico sul backend.

3.1.4 Applicazione mobile

L'ultimo componente dell'architettura è l'applicazione mobile. Essa è sviluppata attraverso il framework Angular2+ e Ionic e al suo interno risiede il bot Azzurra. Oltre al bot Azzurra esisto altre due sezioni, la sezione questionario e la sezione profilo.

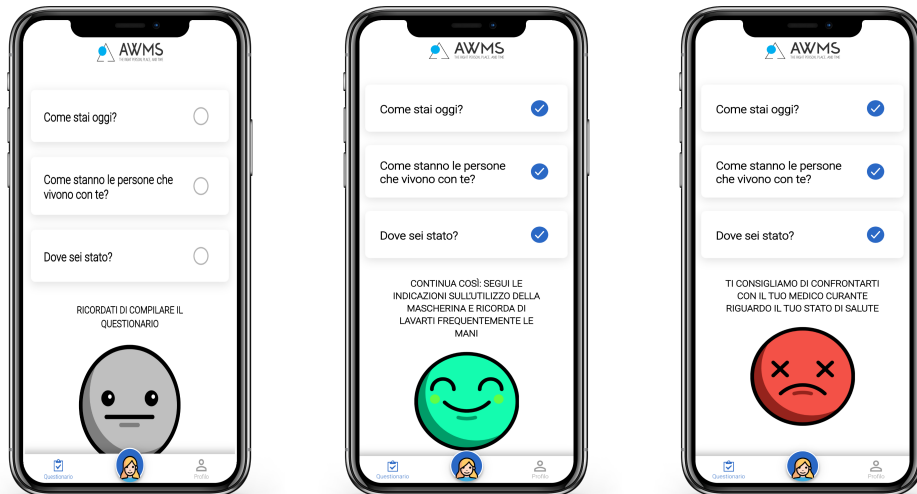


Figura 3.3: Sezione Questionario

La Figura 3.3 mostra la sezione Questionario nei suoi 3 possibili stati. In questa sezione viene richiesto di compilare quotidianamente un questionario in cui vengono poste domande sulla propria salute che, dai dati raccolti per ogni lavoratore, l'applicazione cerca di capire se all'interno dell'azienda ci sia pericolo di contagio del virus COVID-19. Nel caso in cui non si è ancora compilato il questionario, viene mostrata una faccina grigia come si può vedere nella prima immagine della Figura 3.3. Se si è compilato il questionario e secondo le risposte date si risulta essere in buona salute, allora l'applicazione mostrerà una faccina verde come si può vedere nella seconda immagine della Figura 3.3. Se si è compilato il questionario e secondo le risposte date si risulta essere a rischio con la propria salute, allora l'applicazione mostrerà una faccina rossa come si può vedere dalla terza immagine della Figura 3.3.

In questa sezione viene richiesto di compilare un questionario dove vengono richiesti se si hanno avuto dei sintomi di malattie come mostra la prima immagine della Figura 3.4. Successivamente viene richiesto se le persone vicino a noi hanno avuto qualche sintomo di malattie come mostrato nella seconda immagine della Figura 3.4. Viene poi richiesto in quelli luoghi si è stati come mostrato nella seconda immagine della Figura 3.4.

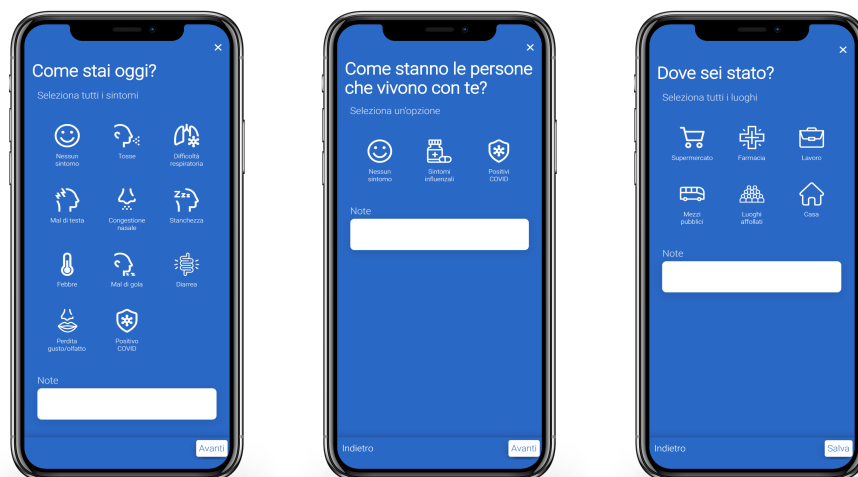


Figura 3.4: Schede del questionario sulla salute

Una volta terminato il questionario, l'applicazione elaborare le risposte date e mostrerà l'esito sulla nostra salute. Nel caso in cui l'esito sia positivo verrà mostrata la prima immagine della Figura 3.5 invece in caso di esito negativo verrà mostrata la seconda immagine della Figura 3.5.



Figura 3.5: Schede dell'esito del questionario sulla salute

Il risultato viene poi riportato anche nella schermata della sezione Questionario. Nella sezione Profilo invece, vengono mostrati i Karma points che sono stati raccolti durante la compilazione del questionario, punti che al momento non danno nessun particolare beneficio ma, in futuro è previsto l'implementazione di una qualche ricompensa. Vengono mostrate le proprie informazioni personali cliccando il tasto Informazioni personali come mostrato in Figura 3.6, e possibile cambiare la password d'accesso cliccando il bottone Gestione password. Cliccando il bottone Istruzione di utilizzo mostrato sempre nella Figura 3.6 e possibile accedere a una breve guida su come utilizzare l'applicazione. Nel bottone Normativa privacy è possibile visionare la normativa sulla tutela della privacy GDPR mentre nel bottone Titolare trattamento viene indicato da chi vengono trattati i dati inseriti.



Figura 3.6: Sezione Profilo



Figura 3.7: Sezione Chat bot Azzurra

Nella sezione Azzurra, è presente la chat bot con Azzurra che attraverso il proprio flow engine, riesce a comprendere i flussi conversazionali ricevuti in input da Azzurra.io. Grazie a ciò il bot Azzurra sa quali risposte e domande fare all'utente umano. Nel capitolo successivo verrà spiegato in modo dettagliato il funzionamento del Flow engine

di Azzurra. Come detto precedentemente la comunicazione con Azzurra.io avviene attraverso WebSocket che permette di tenere aggiornati i flussi conversazionali ricevuti da Azzurra.io nel caso in cui subiscano modifiche.

3.2 Operazioni

Nella seguente sezione verranno descritte le principali operazioni tra le varie componenti dell'architettura.

3.2.1 Creazione di una connessione attraverso WebSocket

Come spiegato in precedenza, la comunicazione tra l'applicazione mobile e Azzurra.io avviene attraverso l'utilizzo di WebSocket. Grazie a ciò si ha un canale di comunicazione a due vie cioè, sia l'applicazione mobile e sia Azzurra.io possono inviare dati o richieste inoltre, grazie a questo tipo di comunicazione, le modifiche ai flussi conversazionali già esistenti o l'aggiunta di nuovi flussi, verranno comunicate all'applicazione mobile in tempo reale, aggiornando perciò i dati posseduti dell'applicazione. Infine utilizzando una connessione tramite WebSocket, risulta essere più efficiente e performante rispetto al pooling perchè il server non viene continuamente contatto da inutili richieste.

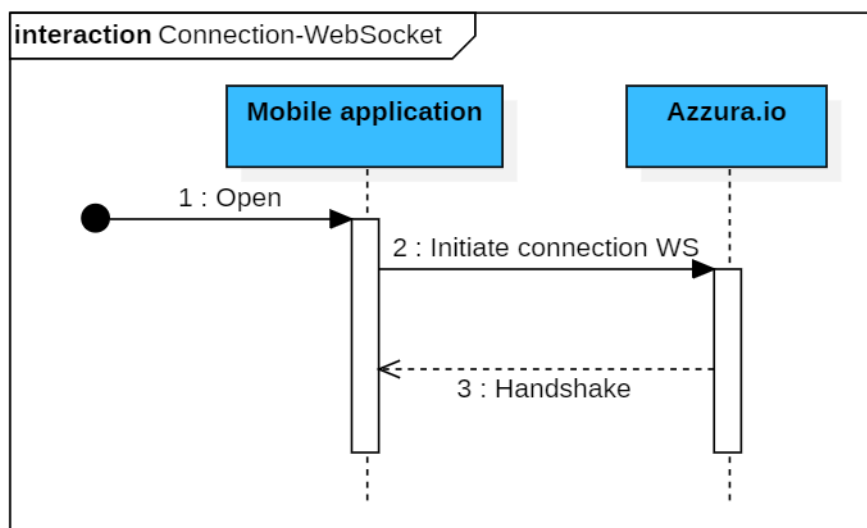


Figura 3.8: Sequence diagram per la creazione di una connessione attraverso WebSocket

Nella Figura 3.8 viene mostrato come avviene la creazione di una connessione tra l'applicazione mobile e Azzurra.io, i passi perciò sono:

1. All'inizio, l'applicazione mobile viene aperta dall'utente, essa cercherà da subito di mettersi in contatto con Azzurra.io creando una connessione;
2. Per avviare una connessione WebSocket, viene inviata una richiesta HTTPS a Azzurra.io(server). Tale richiesta HTTPS ha la particolarità che negli headers dell'intestazione viene specificata un'operazione di tipo Upgrade che indica che l'applicazione mobile (client) vuole aggiornare la connessione ad un protocollo

diverso, in questo caso a WebSocket. Questo tipo di operazione prende il nome di WebSocket handshake request.

```
GET /mychat HTTPS/1.1
Host: server.AzzurraIo.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: 32ndfsMjnQiZXBijAf0iPni==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 13
Origin: http://AzzurraIo.com
```

Il codice soprastante rappresenta un esempio di WebSocket handshake request inviata dal client al server. Il client inserisce una stringa casuale codifica in base64 nel campo Sec-WebSocket-Key alla quelle viene poi aggiunta una stringa fissa.

- Se il server, in questo caso Azzurra.io supporta la connessione tramite WebSocket allora rispondere mettendo nel campo Sec-WebSocket-Accept la risposta cioè, l'hash della stringa contenuta in Sec-WebSocket-Key utilizzando la funzione di hashing SHA-1. Infine, viene tutto codificato in base64. Una volta arrivata la risposta al client, esso controllerà se la risposta contiene la stringa corretta. Tutte queste operazioni hanno lo scopo di evitare di aprire più connessioni multiple ma non da nessuna garanzia di autenticazione, tale garanzia è data dal utilizzo del HTTPS. Nel seguente codice viene riportato un esempio di risposta da parte del server.

```
HTTPS/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: fPmrc0slUIUYIuyY2HaGWk=
Sec-WebSocket-Protocol: chat
```

3.2.2 Recupero di un flusso conversazionale

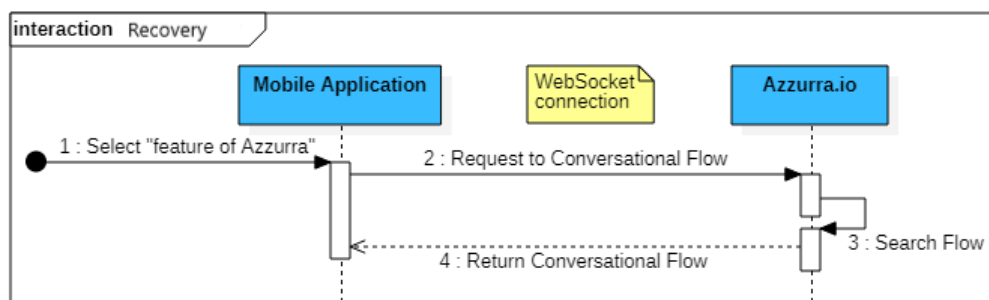


Figura 3.9: Sequence diagram per il recupero di un flusso conversazionale

Il bot Azzurra per poter funzionare ha bisogno di almeno un flusso conversazionale che grazie alla propria struttura, permette al flow engine di Azzurra di comprendere il flusso e capire quali messaggi deve far visualizzare ad Azzurra nella chat con l'utente

umano. I flussi si trovano nel database dedicato di Azzurra.io. Quando viene richiesto di fare un'operazione o si inizia per la prima volta un'interazione con Azzurra, essa per sapere cosa fare ha bisogno di uno specifico flusso di conversazione, nel caso in cui sia la prima interazione con l'utente, richiede il cosiddetto MainFlow dove vengono riportate quali funzionalità Azzurra può offrire. Se invece si è nel caso in cui l'utente richiede una specifica funzionalità ad esempio richiedere la visualizzazione del piano di lavoro, Azzurra dovrà richiedere il flow dedicato alla specifica funzionalità. Come mostrato nella Figura 3.9 si hanno i seguenti passi:

1. L'utente interagisce con il bot Azzurra e richiede una funzionalità oppure l'utente interagisce con Azzurra per la prima volta;
2. Il bot Azzurra tramite una connessione aperta precedentemente chiede a Azzurra.io il corretto flow per poter soddisfare le richieste dell'utente. Si sottolinea che ogni flow ha un codice identificativo che lo identifica e Azzurra sa sempre qual'è il codice identificativo del flusso di cui ha bisogno;
3. Azzurra.io cerca nel proprio database se contiene il flow richiesto;
4. Se la ricerca da esito positivo allora Azzurra.io ritornerà il flusso che verrà eseguito dal flow engine di Azzurra la quale proseguirà con la conversazione. Se invece non lo trova la conversazione si interrompe mostrando nella chat un messaggio di errore.

3.2.3 Richiesta e invio di dati

Durante l'esecuzione di una conversazione è molto probabile che Azzurra abbia bisogno di far visualizzare delle informazioni richieste dall'utente ad esempio il suo orario di lavoro, queste informazioni però non sono salvate né nell'applicazione mobile né in Azzurra.io ma nel backend. La Figura 3.10 mostra come Azzurra ottiene i dati richiesti:

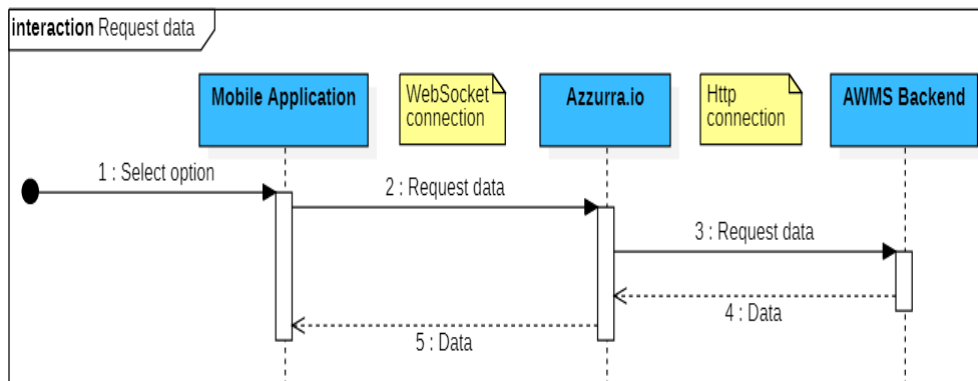


Figura 3.10: Sequence diagram per il recupero dei dati sul lavoratore

1. L'utente chiede una qualche operazione ad Azzurra che necessita contattare il backend per ottenere i dati che l'utente vuole;
2. L'applicazione chiede quindi i dati necessari a Azzurra.io che si prende carico della richiesta;

3. Azzurra.io contatta attraverso una richiesta HTTP, per richiedere i dati richiesti da Azzurra;
4. Se la richiesta va a buon fine il backend ritornerà i dati ad Azzurra.io che li ritornerà a sua volta ad Azzurra.

Per l'invio dei dati invece, ciò l'utente inserisce dei dati che devono essere salvati sul database del backend, ad esempio l'utente inserisce una nuova assenza, il procedimento sarà analogo alla richiesta di dati solo alla fine non verranno ritornati i dati ma l'esito dell'operazione effettuata.

3.2.4 Gestione notifiche push

Nel caso in cui il project manager abbia bisogno di mandare una comunicazione a uno o più lavoratori, e stata implementata la possibilità di inviare delle notifiche contenenti la comunicazione.

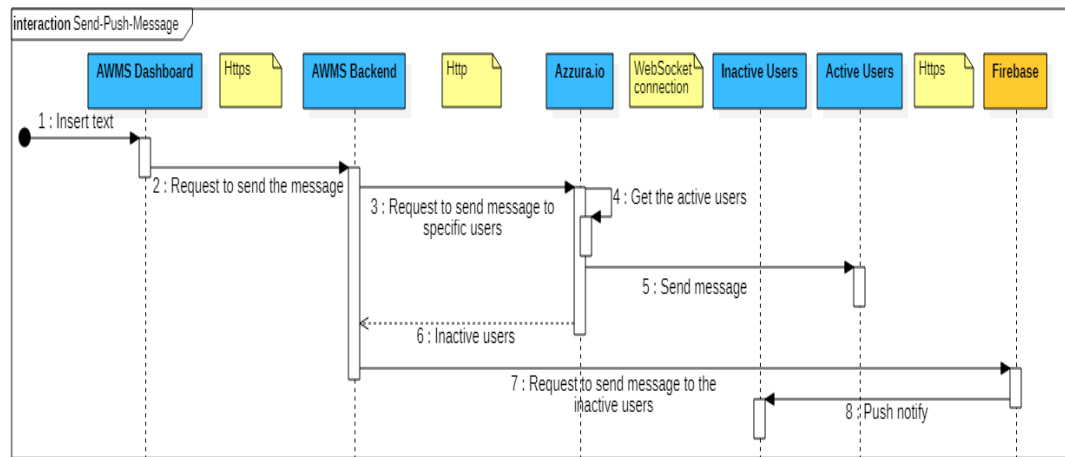


Figura 3.11: Sequence diagram per l'invio di notifiche push

Nella Figura 3.11 viene mostrato come avviene l'invio delle notifiche, di seguito ne vengono descritti i passi:

1. Il project manager crea il messaggio da inviare attraverso l'interazione con la dashboard di AWMS;
2. La dashboard invia al backend il messaggio da inviare e a quali utenti registrati nel sistema, mandare il messaggio, viene perciò mandato oltre al messaggio anche i codici identificativi dei destinatari;
3. Il backend si connette ad Azzurra.io e gli invia il messaggio e la lista degli utenti a cui deve mandare la notifica. Quindi l'invio della notifica viene delegato a Azzurra.io;
4. Azzurra.io controlla sulla sua tabella User Map, quali utenti che devono ricevere il messaggio sono attualmente connessi al socket, oppure invece non lo sono. Si specifica inoltre che vengono considerati utenti connessi o attivi tutti gli utenti

che sono connessi e che hanno l'applicazione in foreground. Gli utenti che sono connessi al socket, ma hanno l'applicazione in background vengono trattati come utenti non connessi;

5. Per gli utenti connessi al socket, Azzurra.io invierà la notifica ai dispositivi connessi in cui verrà visualizzata sotto forma di messaggio da parte di Azzurra;
6. Nel caso in cui ci siano alcuni utenti non connessi a Azzurra.io tra i destinatari del messaggio, Azzurra-io effettua una richiesta HTTP ad AWMS Backend, specificando gli utenti che non è riuscita a contattare, e il messaggio che avrebbe dovuto recapitare;
7. Il backend una volta ricevuta la lista degli utenti a cui non è stato possibile mandare la notifica, contatterà attraverso un API⁸ di Firebase, quest'ultimo per richiedere l'invio della notifica ai utenti della lista appena ricevuta. Si specifica che Firebase sarà in grado di farlo solo se i dispositivi dei destinatari si sono sottoscritti al servizio di Firebase per la ricezione di notifiche push;
8. Firebase ricevuta la lista d'utenti a cui inviare la notifica e il testo del messaggio, invierà la notifica push ai destinatari.

Capitolo 4

Azzurra Flow Engine

Nel seguente capitolo verrà illustrato prima di tutto la struttura di un flusso conversazionale e successivamente il funzionamento del Azzurra Flow Engine la conseguente generazione dei messaggi da parte del bot Azzurra e dell'utente umano

4.1 Scopo

Un elemento cardine dell'architettura di **Azzurra.flow** è **Azzurra Engine**. È un motore conversazionale in grado di ricevere in input dei flussi di conversazione i quali sono implementati attraverso file JSON, che risiedono nel database di **Azzurra.io**. Essi vengono mandati in input a **Azzurra Engine** quando il bot **Azzurra** ne fa richiesta. Questi file sono codificati secondo una certa sintassi fatta dai cosiddetti blocchi conversazionali i quali verranno illustrati in seguito. Tornando su **Azzurra Engine** come detto, riceve il file di conversazione in JSON in input e grazie ai metodi che ha disposizione è in grado di interpretare i file JSON ricevuti e generare i messaggi che il bot **Azzurra** deve fare visualizzare all'utente nella chat.

4.2 Flussi di conversazione

I flussi di conversazione sono degli elementi fondamentali per la conversazione tra il bot e l'utente umano. Essi sono delle configurazioni JSON, dove ogni configurazione JSON contiene un flusso di conversazione, e ogni flusso è un possibile ramo di conversazione che può essere fatto tra il bot Azzurra e l'utente umano. Ogni configurazione JSON contiene perciò dei particolari comandi che permettono al bot di sapere quali messaggi deve mostrare all'utente umano e come comportarsi in base alle sue scelte. Ogni configurazione JSON ha un **id** che contiene un codice univoco in modo tale da poter identificare ogni flusso di conversazione. Inoltre, l'esecuzione dei flussi di conversazione prevede che all'inizio ci sia l'esecuzione di un cosiddetto main flow, in modo simile a come avviene per i programmi software, cioè c'è una funzione detta main che viene eseguita per prima all'avvio del programma. Per indicare quale tra l'insieme dei flussi sia il main si utilizza il campo **isMainFlow** dandogli il valore **true**. Oltre a questi campi esistono altri campi che sono:

- * **Shortcuts** "shortcuts";

- * Configurazione "config";
- * Blocchi per la conversazione "blocks".

Tutti e tre verranno illustrati nelle seguenti sottosezioni.

4.2.1 Shortcuts “shortcuts”

Il campo **shortcuts** è il campo dedicato per le cosiddette "scorciatoie" cioè, nelle funzionalità che il bot offre all'utente c'è anche la possibilità di visualizzare un menu dove vengono mostrate tutte le funzionalità offerte dal bot e scegliere direttamente quelle eseguire in ogni momento.

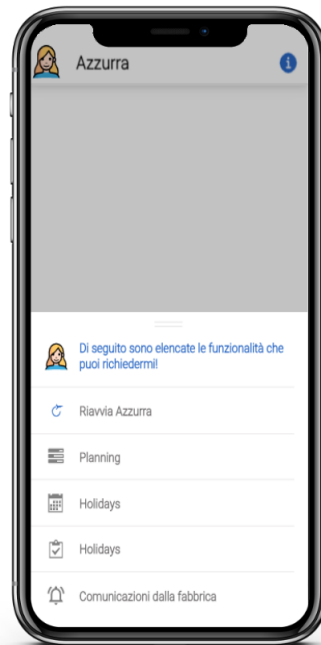


Figura 4.1: Menu contenente le shortcuts disponibili

Il campo **shortcuts** viene indicato nel file con la keyword **shortcuts** e al suo interno contiene i seguenti campi:

- * **text:** È un campo che può essere di tipo *string* contenente il testo da far visualizzare nel bottone della scorciatoia all'utente, oppure un oggetto che contiene un attributo per ogni lingua disponibile. Il testo nella lingua di default (italiano) è contenuto nell'attributo "default" ;
- * **flowId:** Contiene l'identificativo del flusso che la scorciatoia permette di far eseguire;
- * **icon:** Per rendere la UI più accattivante è possibile aggiungere al bottone dedicato alla scorciatoia, delle icone per ogni scorciatoia.

4.2.2 Configurazione "config"

Il campo **config** permette di indicare attraverso il campo **startBlockId** quale blocco di conversazione del flusso deve essere eseguito per primo, per tale campo ci sarà il codice identificativo del primo blocco da eseguire. Il campo **configurazione** viene indicato nel file con la keyword **config**.

4.2.3 Blocchi per la conversazione

Il campo **Blocchi** indicato nel file con la keyword **blocks** contiene tutti i blocchi per la conversazione i quali indicano quali messaggi devono essere mostrati e quali operazioni eseguire a seconda delle scelte inserite dell'utente umano. I blocchi utilizzati per la conversazione si differenziano tra loro dal tipo di blocco a cui appartengono. Ogni tipo ha proprie caratteristiche uniche ma anche delle caratteristiche comuni questo perché tutti i tipi di blocchi per la conversazione ereditano da un tipo padre detto **BLOCK**. I tipi figli di **Block** sono i seguenti:

- * **ASK**;
- * **SAY**;
- * **IF**;
- * **PROC**;
- * **JUMP**;
- * **CALLFUNC**.

Di seguito verrà illustrata la struttura del tipo e il ruolo di **BLOCK** e dei suoi figli.

BLOCK

Il blocco **BLOCK** come detto è il blocco di conversazione attraverso il quale tutti blocchi ereditano delle caratteristiche comuni della loro struttura, di fatto il blocco **BLOCK** non può essere utilizzato è quindi può essere paragonato a una classe astratta nell'ambito della programmazione ad oggetti, dove vengono definite delle caratteristiche della classe ma non può essere istanziata, perciò può essere solo ereditata dai suoi figli, che diventano classi concrete istanziabili.

Ha la seguente struttura:

- * **id**: È un campo di tipo *string* il quale identifica univocamente il blocco tra un insieme di blocchi di conversazione;
- * **type**: Questo campo indica il tipo di blocco che come detto può essere di tipo **ASK** o **SAY** o **IF** o **PROC** o **JUMP** oppure **CALLFUNC**;
- * **text**: È un campo che può essere di tipo *string* contenente il testo da far visualizzare all'utente, oppure un oggetto che contiene un attributo per ogni lingua disponibile contenente del testo nella corrispondente lingua, e un attributo default che contiene il testo di default;
- * **variations**: Anch'esso è un tipo *string* che contiene uno o più testi alternati al principale rappresentato dal campo **text**. Il funzionamento prevede che randomicamente il testo da mostrare all'utente non sarà quello principale ma

uno delle alternative contenuto all'interno di **variations**, verrà perciò scelto in modo casuale, uno dei testi a disposizione;

- * **target**: Questo campo contiene l'id del prossimo blocco di conversazione da eseguire;
- * **variable**: Questo campo indica il nome della variabile su cui salvare eventuali valori di input inseriti dall'utente;
Perciò, il flow engine attraverso i suoi metodi, ha la capacità di salvare tutte le scelte fatte dell'utente, memorizzandole nelle variabili indicate nel campo **variable**.
- * **widget**: Indica il tipo di oggetto grafico detto **Widget** che deve essere utilizzato a supporto del blocco, esistono i seguenti tipi di **Widget**:
 - **BUTTONS**;
 - **ITEMS**;
 - **PICKER**;
 - **TIMEPICKER**;
 - **DATEPICKER**;
 - **CALENDAR**;
 - **QRSCANNER**.

Più avanti verranno illustrati in dettaglio;

- * **widgetOptions**: Permette di aggiungere delle opzioni in più al **Widget** per esempio permette di indicare del testo all'interno dei **Widget** oppure indicare il valore minimo accettabile.

ASK

Il blocco di conversazione **ASK** ha la funzione di mostrare all'utente una serie di opzioni disponibili e chiedere quali tra queste vuole eseguire. Quindi mostra le opzioni definite precedentemente all'utente, rimane in attesa di una risposta dell'utente e infine esegue la scelta effettuata dall'utente.

Oltre ai campi del tipo **BLOCK** ha i seguenti campi in più:

- * **category**: Il blocco **ASK** è ulteriormente distinguibile in **ASK** o in **MENU** che si differenziano nel seguente aspetto:
nel caso sia di categoria **ASK** tutte le opzioni disponibili portano a un blocco di conversazione successivo diverso mentre per **MENU** tutte le opzioni portano tutte allo stesso blocco;
- * **source**: Parametro che contiene il nome di una variabile a cui fare riferimento per prendere i dati da utilizzare, se non si fa riferimento a nessuna variabile allora va settato a **NULL**;
- * **items**: Questo campo contiene un array d'oggetti di tipo *BlockItem* che rappresentano le possibili scelte che può fare l'utente, essi graficamente vengono rappresentati come dei pulsanti;

- * **sourceType**: Parametro che indica la modalità di utilizzo della variabile contenuta nel campo **source**; Ci sono solo due modalità disponibili:
 - **LIST**: In questo caso la variabile contenuta in **source** viene ignorata e vengono presi tutti gli oggetti di tipo *BlockItem* contenuti nel campo **items** che vengono trasformati in bottoni da far visualizzare a video;
 - **VARIABLE**: In questo caso tutto ciò che è contenuto nella variabile del campo **source** viene trasformato in bottoni da far visualizzare a video, per poterlo fare devono essere formattati in una struttura del tipo chiave valore.

SAY

Il blocco di conversazione **SAY** ha la funzione di mostrare all'utente un messaggio a video attraverso il quale si comunica l'esito della operazione precedente e il risultato da essa ricavato. Perciò l'utente richiede l'esecuzione di una qualche operazione, viene eseguita e una volta conclusa il bot risponderà all'utente con il risultato ricavato precedentemente.

Oltre a campi del tipo **BLOCK** ha il seguente campo in più:

- * **attachment**: Campo che contiene un array d'oggetti di tipo *BlockAttachment* che permettono di allegare immagini o file PDF.

Inoltre sono presenti i campi **items**, **source** e **sourceType**, con analogo funzionamento del blocco **ASK** per inserire eventuali bottoni che aprono schede o link contenenti il risultato richiesto.

IF

Il blocco di conversazione **IF** ha la funzione di verificare se una o più condizioni sono rispettate. Perciò verifica se le condizioni sono soddisfatte se si farà un certo tipo di operazioni previste, se invece non sono soddisfatte si eseguiranno altre operazioni.

Oltre a campi del tipo **BLOCK** ha i seguenti campi in più:

- * **conditions**: Contiene una o più condizioni che devono essere verificate;
- * **trueBlockTarget**: Indica il blocco successivo da eseguire se le condizioni sono soddisfatte;
- * **falseBlockTarget**: Indica il blocco successivo da eseguire se le condizioni non sono soddisfatte.

PROC

Il blocco di conversazione **PROC** permette di eseguire delle operazioni sulle variabili conversazionali quali assegnazione o trasformazione di dato. Ad esempio, permette di riordinare i dati ricevuti dal server in modo da poter essere utilizzati dai **source** con **sourceType** uguale a **VARIABLE**.

Oltre a campi del tipo **BLOCK** ha il seguente campo in più:

- * **expressions**: Contiene le espressioni da eseguire, ad esempio, per la formattazione dei dati. Ha i seguenti campi:
 - **var**: Contiene il nome della variabile dove viene salvato il risultato della formattazione;

- **type**: Indica il tipo di formattazione che si vuole applicare, al momento c'è solo una formattazione disponibile:
 - * **reduce to textvalue**: Permette di riordinare i vari valori che si hanno in una struttura chiave valore.
- **args**: Contiene un espressione in **Handlebars**, un linguaggio di templating utilizzato per costruire template in HTML con dei cosiddetti segnaposto che verranno poi valorizzati con dei valori, utilizzando delle keyword del linguaggio, in modo da ottenere delle componenti in HTML da mostrare come messaggio.

JUMP

Il blocco di conversazione **JUMP** permette di cambiare il flusso di conversazione ed eseguirne uno altro. In termini tecnici si passa da un JSON di configurazione ad un altro dove ogni configurazione JSON contiene un specifico flusso di esecuzione, grazie a **JUMP** si può "saltare" da un flusso di conversazione a un altro.

Nel campo **target** non viene indicato l'id del blocco successivo ma, l'id del flow che si vuole eseguire.

CALLFUNC

Il blocco di conversazione **CALLFUNC** è il blocco attraverso il quale, il bot (la mobile App) può richiedere l'esecuzione di chiamate ad API (interne o esterne). Attraverso un WebSocket, che mantiene una connessione tra il bot e Azzurra.io, quest'ultima richiama, a sua volta, delle API di AWMS (o esterne ad AWMS) per ottenere i dati richiesti dell'utente o per salvare dati.

Oltre ai campi del tipo **BLOCK** ha il seguente campo in più:

- * **payload**: Campo che contiene l'intestazione e il corpo della richiesta verso Azzurra.io; Contiene i seguenti campi:
 - **type**: Indica se la chiamata è verso Azzurra.io attraverso il valore **int** oppure verso un servizio esterno con il valore **ext**.

Se la chiamata è di tipo **int** ha la seguente struttura:

- **route**: Indica il metodo di Azzurra.io da richiamare ;
- **body**: Contiene il corpo della richiesta, nello specifico un template costruito da **Handlebars** che verrà idratato da Azzurra.io nel caso sia una richiesta di dati o dal bot nel caso in cui debba inviare dei dati da salvare.

Se la chiamata è di tipo **ext** ha la seguente struttura:

- **config**: Contiene la struttura di una chiamata HTTP. Ha i seguenti campi:
 - * **url**: Contiene l'indirizzo URL del servizio esterno a cui fare richiesta;
 - * **method**: Se la richiesta è di tipo **GET** o **POST**;
 - * **headers**: Contiene l'intestazione per la richiesta HTTP;
 - * **params**: Contiene le variabili necessarie per la chiamata, questo campo viene usato solo se la richiesta è di tipo **GET**;

- * **data**: Analogo al campo **params** solo che viene usato dai metodi POST.
- * **var**: Indica il nome della variabile dove salvare il risultato della richiesta;
- * **failureBlockTarget**: Indica il blocco successivo da eseguire se la richiesta non va a buon fine;
- * **successBlockTarget**: Indica il blocco successivo da eseguire se la richiesta va a buon fine.

4.2.4 Oggetti ausiliari

Come detto nella sezione precedente questi oggetti vengono definiti per poterli utilizzarli all'interno dei blocchi per svolgere un'azione di supporto ai blocchi affinché si possa raggiungere ciò per cui sono stati realizzati i blocchi di conversazione stessi.

Di seguito vengono indicate tutte le classi degli oggetti ausiliari disponibili.

Widget

È un oggetto che a seconda del tipo permette di realizzare delle componenti grafiche, esso viene utilizzato per richiedere delle azioni da parte dell'utente umano. Ha i seguenti tipi:

- * **BUTTONS**: Genera dei bottoni arrotondati;
- * **ITEMS**: Genera dei bottoni quadrati;
- * **PICKER**: Genera attraverso l'ion-picker di Ionic una finestra di dialogo dove si può selezionare una opzione tra quelle proposte;
- * **TIMEPICKER**: Analogo al **PICKER** solo che le opzioni da scegliere e l'orario che si vuole selezionare;
- * **DATEPICKER**: Analogo al **PICKER** solo che le opzioni da scegliere e la data che si vuole selezionare;
- * **CALENDAR**: Fa comparire un calendario grazie all'utilizzo del plugin Calendar per Ionic;
- * **QRSCANNER**: Permette di accedere alla fotocamera (solo se si hanno i permessi) e leggere i codici QR-code tutto ciò grazie al plugin QR Scanner per Ionic.

BlockItem

Questo oggetto rappresenta una possibile scelta che può fare l'utente, graficamente viene rappresentato come un bottone cliccabile dall'utente. Ha la seguente struttura:

- * **text**: Contiene l'etichetta che viene visualizzata sul bottone;
- * **target**: Contiene l'id del prossimo blocco da eseguire.

BlockAttachment

L'oggetto in esame permette di allegare immagini o file PDF da mostrare all'utente.
Ha la seguente struttura:

- * **id**: Contiene un codice univoco che identifica ogni *BlockAttachment*;
- * **type**: Indica se contiene un PDF o una immagine, nel caso di un'immagine indica se è in formato JPG o in JPEG oppure in PNG.

Capitolo 5

Progettazione e codifica

Breve introduzione al capitolo

5.1 Tecnologie e strumenti

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati.

Tecnologia 1

Descrizione Tecnologia 1.

Tecnologia 2

Descrizione Tecnologia 2

5.2 Ciclo di vita del software

5.3 Progettazione

Namespace 1

Descrizione namespace 1.

Classe 1: Descrizione classe 1

Classe 2: Descrizione classe 2

5.4 Design Pattern utilizzati

5.5 Codifica

Capitolo 6

Verifica e validazione

Capitolo 7

Conclusioni

7.1 Consuntivo finale

7.2 Raggiungimento degli obiettivi

7.3 Conoscenze acquisite

7.4 Valutazione personale

Appendice A

Appendice A

Citazione

Autore della citazione

Glossario

API in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 35

Acronimi

API Application Program Interface. 14, 23

Bibliografia