

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

Federico Perin: *Analisi, progettazione e sviluppo di un motore conversazionale per una piattaforma di gestione della forza lavoro*, Tesi di laurea triennale, © Settembre 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 obiettivo, 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 fornитоми 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 nell’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	2
1.4 Organizzazione del testo	3
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	6
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	9
2.7.1 Strumenti	9
2.7.2 Tecnologie	11
2.8 Motivazioni personali	12
3 Archittettura del sistema AWMS	13
3.1 Descrizione	13
3.1.1 AWMS Dashboard	13
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 Cos'è	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 "blocks"	27
4.2.4 Oggetti ausiliari	31

4.3	Funzionamento di Azzurra Flow Engine	34
4.3.1	Messaggio del bot Azzurra	34
4.3.2	Messaggio dell'utente umano	39
5	Flussi conversazionali prodotti	41
5.1	Analisi dei requisiti	41
5.1.1	Descrizione del problema	41
5.1.2	Requisiti	41
5.2	Progettazione	43
5.2.1	Gestione delle prenotazioni dei posti	43
5.2.2	Visualizzazione della pianificazione	47
5.3	Codifica	48
5.4	Risultati	49
6	Verifica e validazione	55
7	Conclusioni	57
7.1	Consuntivo finale	57
7.2	Raggiungimento degli obiettivi	57
7.3	Conoscenze acquisite	57
7.4	Valutazione personale	57
A	Appendice A	59
	Acronimi e abbreviazioni	61
	Glossario	63
	Bibliografia	65

ELENCO DELLE FIGURE

1.1	Logo di AzzurroDigitale	1
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	21
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
4.2	Esempio di messaggio prodotto da un blocco di tipo ASK	28
4.3	DEsempio di messaggio prodotto da un blocco di tipo SAY	29
4.4	Rappresentazione grafica dei buttons	31
4.5	Rappresentazione grafica degli items	32
4.6	Rappresentazione grafica del picker	32
4.7	Rappresentazione grafica del time picker	32
4.8	Rappresentazione grafica del date picker	33
4.9	Rappresentazione grafica del QR scanner	33
4.10	Rappresentazione grafica del BlockItem	34
4.11	Diagramma di sequenza del processo di generazione del messaggio del bot Azzurra	35
4.12	Diagramma di sequenza del processo di generazione del messaggio del bot Azzurra	39
5.1	Diagramma per l'inserimento di una nuova prenotazione del flusso DeskBooking	44
5.2	Diagramma per la visualizzazione delle prenotazioni del flusso DeskBooking	46
5.3	Diagramma per lo scansionamento del QR code del flusso DeskBooking	46
5.4	Diagramma per l'inserimento di una nuova prenotazione del flusso DeskBooking	47
5.5	Prenotazione	50
5.6	Prenotazione	51
5.7	Prenotazione	52
5.8	Prenotazione	53

ELENCO DELLE TABELLE

2.1	Tabella del tracciamento dei requisiti qualitativi	9
5.1	Tabella del tracciamento dei requisiti	42
5.2	Tabella del tracciamento dei requisiti	43

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^[g]che grazie a una forte attività collaborazione, fu sviluppata una piattaforma per la gestione degli operai denominata Advanced Workforce Management System (AWMS)^[g], che tutt'ora continua a ricevere miglioramenti e a crescere. Dopo il successo ottenuto con la collaborazione con Electrolux², 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 AWMS².

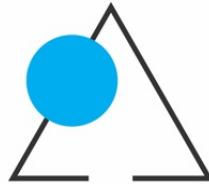


Figura 1.2: Logo di AWMS

AWMS³ è una soluzione software che utilizza algoritmi di Machine learning^[g], per risolvere uno dei problemi cardine di un Plant manager^[g] 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 dei database interni all'azienda che ha acquistato la soluzione, 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. AWMS⁴ 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 AWMS⁵ si ha uno strumento che supporta i *team leader* o i *Plant manager*² 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^[g] 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;
- * Poder effettuare prenotazioni di un posto in una sala riunioni e visualizzare le proprie prenotazioni, inoltre utilizzare un scannerizzatore QR-CODE^[g] 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 conversazionali 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 AWMS⁶ tutto attraverso una comunicazione tramite WebSocket^[g].

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 AWMS⁷ 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.

2 | LO STAGE

Nel seguente capitolo verrà descritto in dettaglio la proposta di stage accettata, 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 AWMS⁸. Perciò era previsto di implementare un'applicazione mobile che potesse comunicare con la piattaforma AWMS⁹, dando un mezzo di supporto al lavoratore di una azienda manifatturiera. All'interno di essa doveva essere implementato una chat bot con un Bot² denominato "**Azzurra**" che offre funzionalità di supporto al lavoratore. All'interno del progetto era anche previsto le implementazioni necessarie per la comunicazione tra AWMS¹⁰ e l'applicazione mobile, quindi la gestione di una connessione WebSocket² 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 AWMS¹¹.

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:

- * Nella 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 End to End (E2E)^[g] per la parte frontend del sistema quindi l'implementare di *test* per la *dashboard* di AWMS¹²;
- * La terza parte era dedicata all'analisi, progettazione e implementazione dei flussi di conversazione per il Bot³ **Azzurra**;
- * La quarta parte era dedicata alla stesura della documentazione per la Solution Design di **Azzurra**;
- * La quinta parte sulle basi che si era imparato nella seconda parte, era previsto lo sviluppo di una test-suite di E2E², con l'obiettivo 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 deiii 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^[g], 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 redirigere 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 AWMS¹³;
- * Aspetti di scalabilità di un flow-engine (concorrenzialità, HA, persistenza/storizziazione 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 <i>test</i> per la <i>dashboard</i> di AWMS ¹⁴ 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 ¹⁵ .
40	03/08/2020 - 07/08/2020	Documentazione delle componenti AWMS ¹⁶ 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, E2E³ con lo scopo di esercitazione. La creazione di E2E⁴ per l'applicazione mobile è stata sposta 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 appresso 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.

3 | ARCHITETTURA DEL SISTEMA AWMS

In questo capitolo verranno descritti tutte le componenti dell'architettura AWMS¹⁷ 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 AWMS¹⁸ e l'applicazione mobile con **Azzurra**.

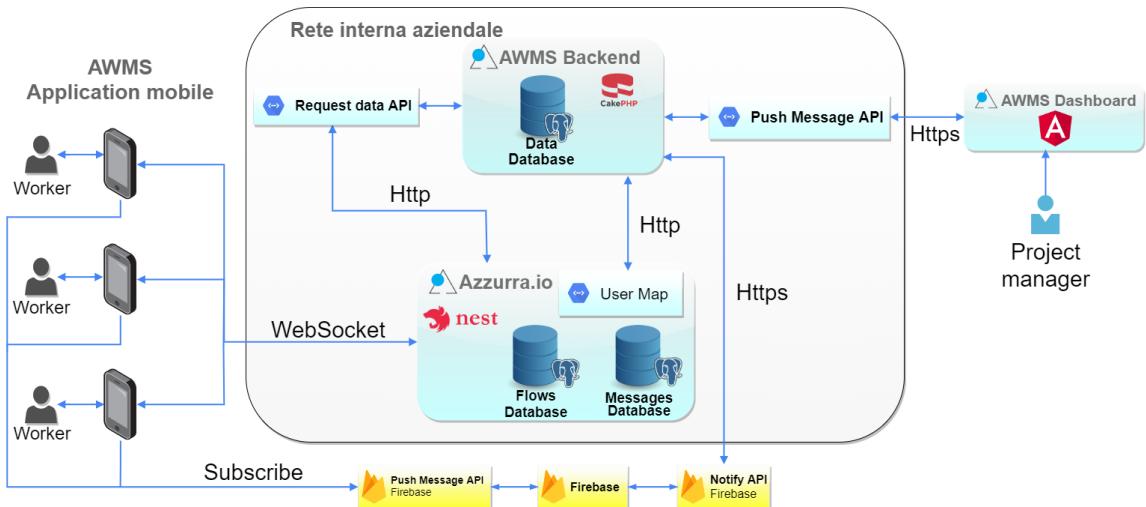


Figura 3.1: Architettura di sistema AWMS

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 AWMS¹⁸ 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, utilizza delle Application Program Interface (API)^g 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 crita

Figura 3.2: Schermata di AWMS Dashboard

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 aperte 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.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 continuare 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 Azzurro.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 dell'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 o 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** esistono altre due sezioni, la sezione questionario e la sezione profilo.

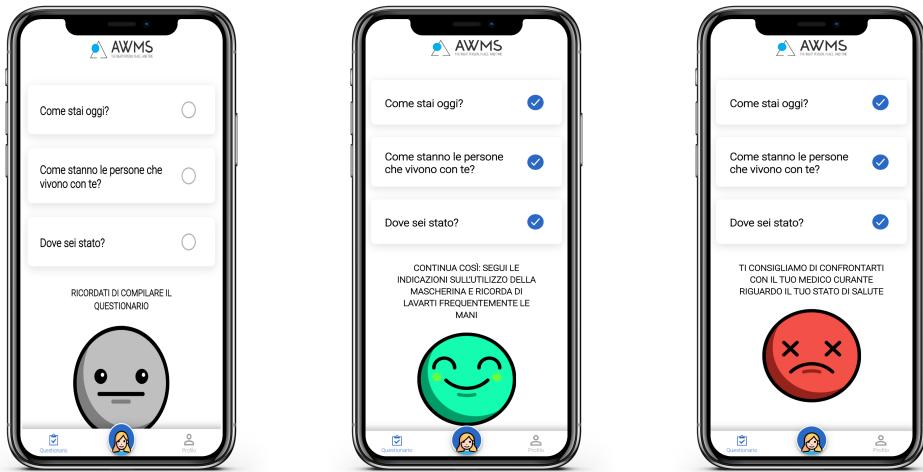


Figura 3.3: Sezione Questionario

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 Istruzioni di utilizzo mostrato sempre nella Figura 3.6 è 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 perché il server non viene continuamente contatto da inutili richieste.

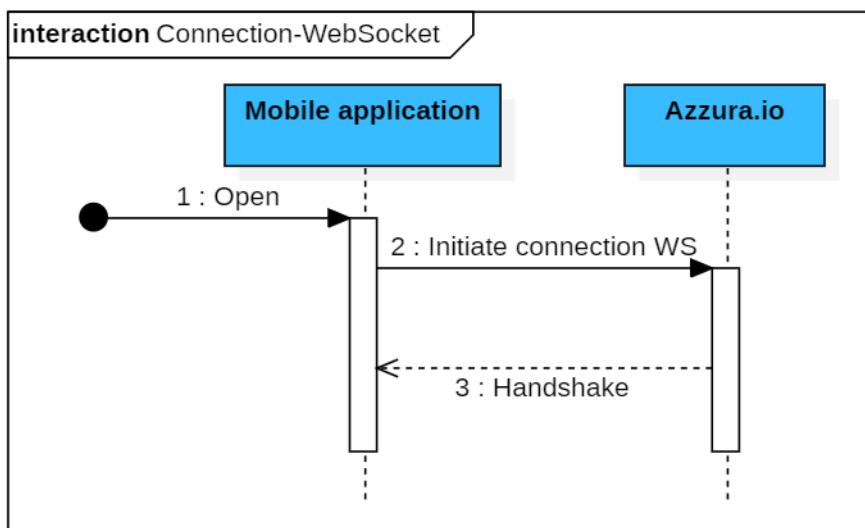


Figura 3.8: Sequence diagram per la creazione di una connessione attraverso WebSocket

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

Il codice soprastrante 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 quale 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. Nel seguente codice viene riportato un esempio di risposta da parte del server.

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

Per garantire una comunicazione sicura contro ascoltatori terzi viene usata una variante del WebSocket detta Secure WebSocket (wss) che utilizza il protocollo HTTPS al posto del protocollo HTTP

3.2.2 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, richiedere 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:

- L'utente interagisce con il bot **Azzurra** e richiede una funzionalità oppure l'utente interagisce con **Azzurra** per la prima volta;

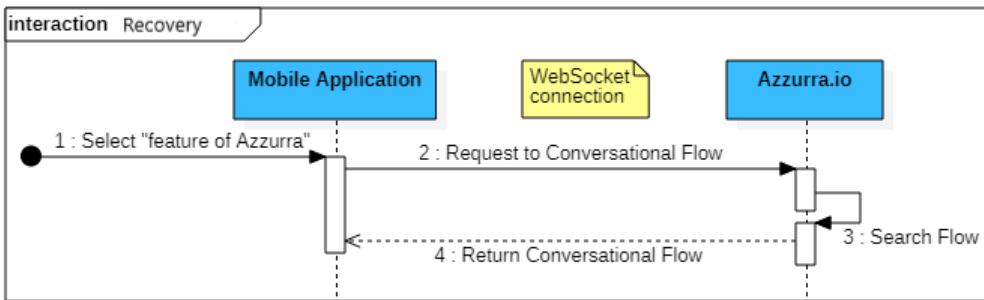


Figura 3.9: Sequence diagram per il recupero di un flusso conversazionale

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** ritirerà 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 ne nell'applicazione mobile né in **Azzurra.io** ma nel backend. La Figura 3.10 mostra come **Azzurra** ottiene i dati

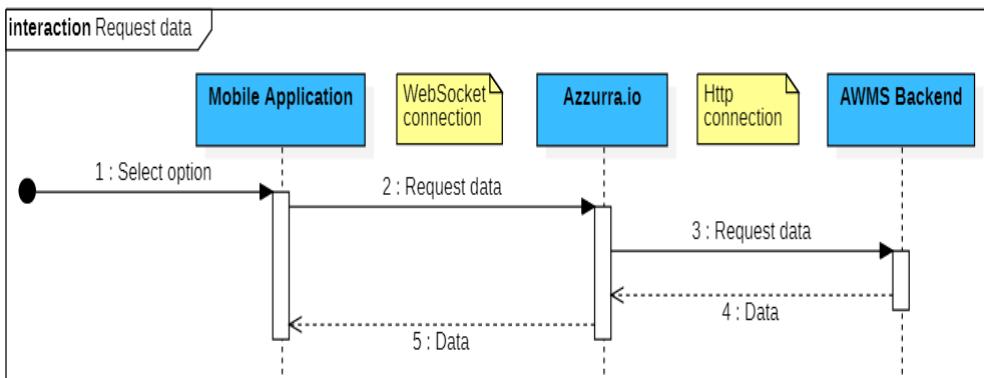


Figura 3.10: Sequence diagram per il recupero dei dati sul lavoratore

richiesti:

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, è stata implementata la possibilità di inviare delle notifiche contenuti la comunicazione.

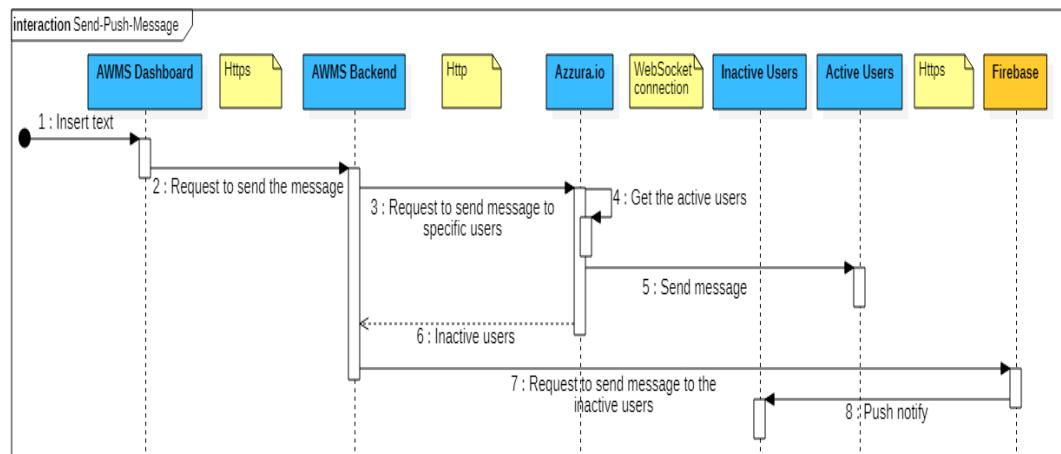


Figura 3.11: Sequence diagram per l'invio di notifiche push

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 devo 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 agli 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.

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 Azzurre e dell'utente umano

4.1 Cos'è

Un elemento cardine dell'architettura di **Azzurra** è **Azzurra Flow Engine**. Esso è un motore conversazionale in grado di ricevere in input flussi conversazionali, implementati attraverso file JSON, che risiedono nel database di **Azzurra.io**. Essi vengono mandati in input a **Azzurra Flow Engine** quando il bot **Azzurra** ne fa richiesta. Questi file sono codificati secondo una certa struttura fatta dai cosiddetti blocchi conversazionali e da altri campi che verranno illustrati in seguito. Tornando su **Azzurra Flow Engine** come scritto, riceve in input il file di conversazione in JSON 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 o conversazionali sono degli elementi fondamentali per la conversazione tra il bot e l'utente umano. Essi sono delle configurazioni in JSON, dove ogni configurazione 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 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 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, tra le 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.

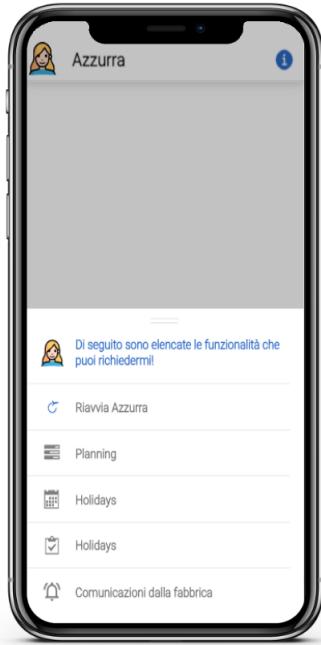


Figura 4.1: Menu contenente le shortcuts disponibili

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* e quindi contiene il testo da far visualizzare nel bottone della scorciatoia all'utente, oppure un oggetto che contiene un attributo per ogni lingua disponibile, dove ogni attributo ha il testo nella lingua straniera che l'attributo rappresenta. 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 "blocks"

Il campo **Blocchi** indicato nel file con la keyword **blocks** contiene tutti i blocchi per la conversazione i quali indicano i messaggi che devono essere mostrati e i passi da eseguire a seconda delle scelte inserite dell'utente umano. I blocchi utilizzati per la conversazione si differenziano tra lo 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 e il ruolo di **BLOCK** e dei suoi figli.

BLOCK

Il blocco **BLOCK** come scritto è il blocco di conversazione attraverso il quale tutti blocchi ereditano delle caratteristiche comuni della sua 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 scritto 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;

- * **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 possibili scelte, rimane in attesa di una risposta dell'utente e infine esegue la scelta effettuata dall'utente passando l'esecuzione al blocco che sa gestire la scelta fatta dall'utente.

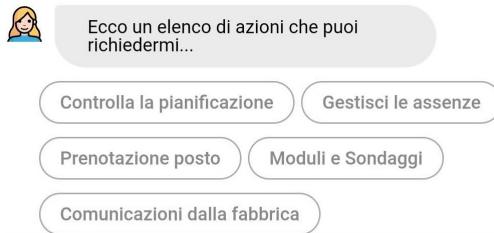


Figura 4.2: Esempio di messaggio prodotto da un blocco di tipo ASK

Oltre a 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 ricavata. 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.



Figura 4.3: DEsempio di messaggio prodotto da un blocco di tipo SAY

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, e in base all'esito verrà scelto il prossimo blocco da eseguire.

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 cioè, assegnazione o trasformazione dei dati. 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 conversazionale ed eseguirne uno altro. In termini tecnici si passa da un JSON di configurazione ad un altro dove ogni configurazione in JSON contiene un specifico flusso di conversazione. Perciò, 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 (l'applicazione mobile) 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 a 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**: Analogico al campo **params** solo se 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 scritto nella sezione precedente questi oggetti vengono definiti per poterli utilizzare 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 button arrotondati;



Figura 4.4: Rappresentazione grafica dei buttons

- * **ITEMS:** Genera dei bottoni quadrati;



Figura 4.5: Rappresentazione grafica degli items

- * **PICKER:** Genera attraverso l'ion-picker di Ionic una finestra di dialogo dove si può selezionare una opzione tra quelle proposte;

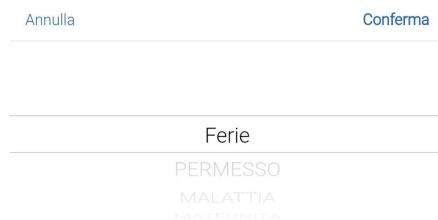


Figura 4.6: Rappresentazione grafica del picker

- * **TIMEPICKER:** Analogico al **PICKER** solo che le opzioni da scegliere e l'orario che si vuole selezionare;



Figura 4.7: Rappresentazione grafica del time picker

- * **DATEPICKER:** Analogo al **PICKER** solo che le opzioni da scegliere è la data che si vuole selezionare;



Figura 4.8: Rappresentazione grafica del date picker

- * **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.

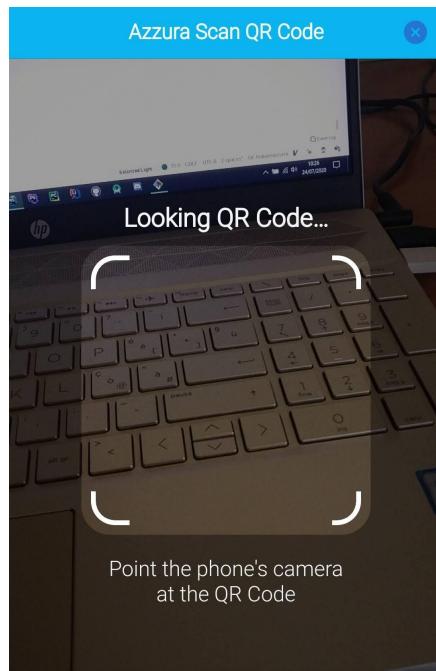


Figura 4.9: Rappresentazione grafica del QR scanner

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:



Figura 4.10: Rappresentazione grafica del BlockItem

- * **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.

4.3 Funzionamento di Azzurra Flow Engine

Come detto il **Azzurra Flow Engine** è l'elemento in grado di interpretare i dati contenuti nelle configurazioni JSON. Grazie a ciò è possibile eseguire il corretto flusso della conversazione e generare i messaggi da mostrare nella *chat* dell'applicazione mobile.

4.3.1 Messaggio del bot Azzurra

Per implementare le funzionalità del **Azzurra Flow Engine** vengono utilizzate le seguenti classi sviluppate in Angular:

- * **FlowService:** Ha i metodi necessari per interpretare le configurazioni JSON e quindi, per poter sapere quale tipo di messaggio deve essere costruito, ricavare i dati necessari per costruire i messaggi e sapere come procedere con la conversazione;
- * **AzzurraService:** Permette principalmente di fare da tramite tra il **FlowService** e il **ChatComponent** per la creazione della conversazione. Oltre a ciò permette di gestire operazioni ad alto livello come il caricamento dei messaggi precedenti nella *chat* e gestire le notifiche che possono arrivare dalla *dashboard*;
- * **ChatComponent:** Si occupa di far visualizzare i messaggi della conversazione;
- * **ChatService:** Ha il compito di creare e gestire i **Widget** da aggiungere al messaggio da mostrare, e inoltre, di ricavarne da essi le risposte dell'utente umano.

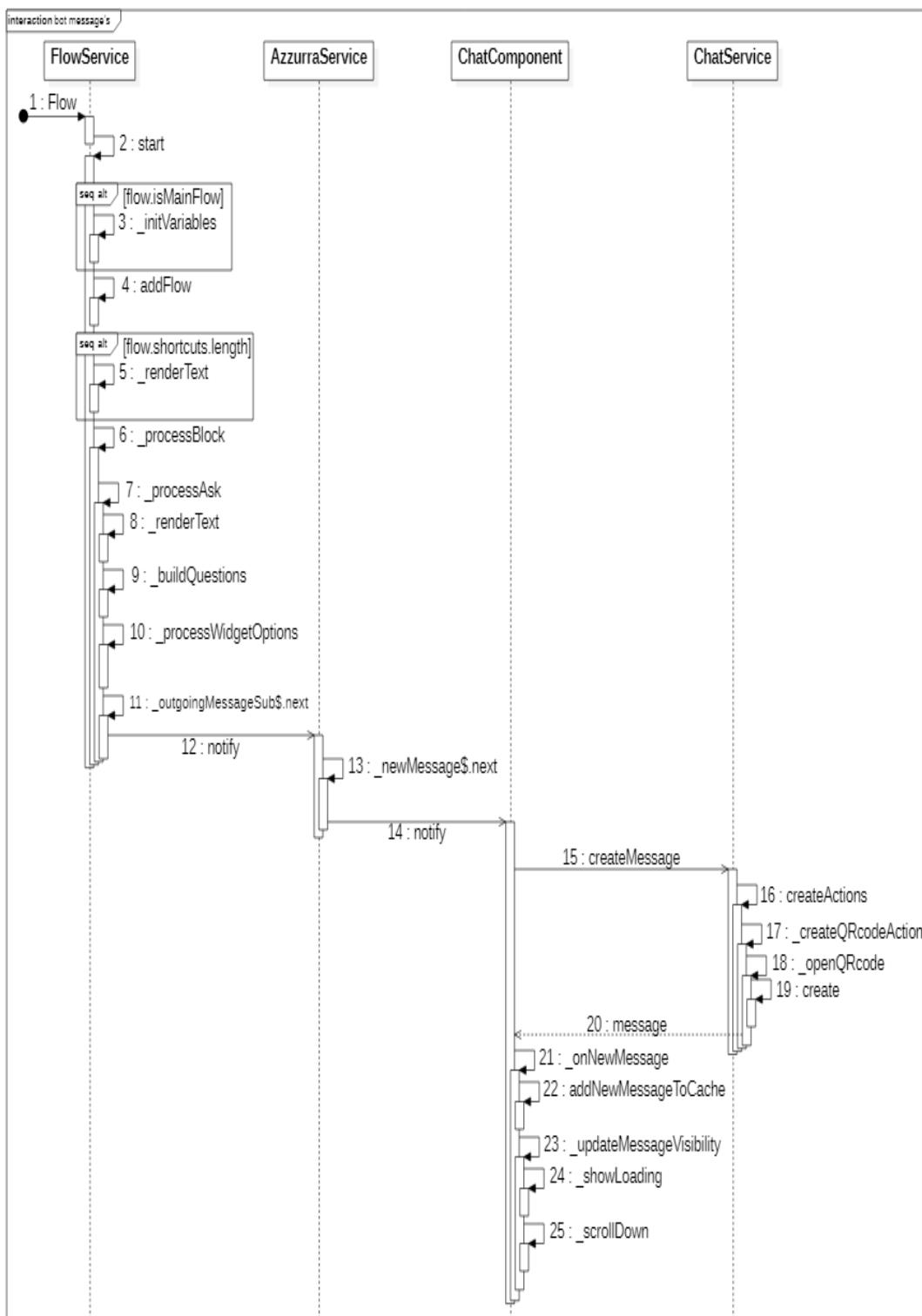


Figura 4.11: Diagramma di sequenza del processo di generazione del messaggio del bot Azzurra

Una volta che si è stabilità la connessione con **Azzurra.io**, attraverso un WebSocket, inizializzato il **Flow Engine** e infine, ricevuto la configurazione JSON contenente il flusso conversazionale, la generazione di un messaggio da parte del bot Azzurra prevede i seguenti passi:

1. Viene inviato all'applicazione mobile il flusso conversazionale richiesto precedentemente;
2. Viene avviato nel **FlowService** il metodo **start()**, il quale riceve in input il flusso da eseguire. Viene verificato se il flusso ricevuto risulta essere il *main flow*, in questo caso vengono inizializzate le variabili conversazionali d'ambiente utilizzate come supporto per la generazione dei messaggi, ad esempio esiste una variabile d'ambiente per indicare il giorno corrente o il formato della data da utilizzare;
3. Una volta impostate le variabile ambiente e usciti dal blocco condizionale, viene salvato il flow da eseguire attraverso il metodo **_addFlow()** che inoltre, imposta come flusso corrente da eseguire il flusso ricevuto in input. Infine, viene controllato se la configurazione definisce delle eventuali **shortcuts**;
4. Se ci sono delle **shortcuts** viene chiamato il metodo **_renderText()** per crearle, esso sarà in grado di interpretare il pezzo di configurazione dove sono definite le proprietà di ogni **shortcuts**;

Una volta finita l'esecuzione del metodo, l'esecuzione torna nel metodo **start()**.

5. Sempre in **start()** viene chiamata l'esecuzione di **_processBlock()** dandogli in input il primo blocco del flusso da eseguire
6. Nel metodo **_processBlock()** viene eseguito il blocco che riceve in input, in questo caso il blocco da eseguire è il primo blocco del flusso come detto nel punto precedente. In questo metodo si verifica innanzitutto se c'è un blocco da eseguire, se non c'è viene emesso un segnale che informa che il flusso di conversazione è terminato, mentre se c'è un blocco allora si imposta questo blocco come quello in esecuzione e si verifica di che tipo è il blocco.

A seconda del tipo del blocco vengono eseguite le seguenti istruzioni:

- * **Caso SAY:** Viene eseguito il metodo **_processSay()** ricevendo in input il blocco corrente. Il metodo prende il valore salvato nel campo **text** e le eventuali **variations** del blocco, generando il testo del messaggio da mostrare attraverso il metodo **_renderText()**. Vengono generati eventuali **BlockItems** attraverso il metodo **_buildSayItems()** il quale controlla il **sourceType** e in base al valore che ha, genera gli **BlockItems**. Tornando in **_processSay()**, se presenti vengono anche eseguite le **widgetOptions** attraverso il metodo **_processWidgetOptions()** e creati gli eventuali **BlockAttachment**. Infine, viene emesso il nuovo messaggio creato per il bot Azzurra e si passa all'esecuzione del prossimo blocco;
- * **Caso ASK:** Viene eseguito il metodo **_processAsk** ricevendo in input il blocco corrente. Il metodo salva il nome della variabile contenuta nel campo **var**, che contiene la risposta dell'utente. Analogamente per quanto accade per il blocco **SAY** viene creato il testo della domanda e attraverso il metodo **_buildQuestions()** vengono creati gli **BlockItems** delle possibili

scelte. Inoltre, vengono processati gli eventuali **widgetOptions**. Infine, viene emesso il nuovo messaggio creato per il bot, il quale rimane in attesa della risposta dell'utente umano quando verrà visualizzato il messaggio;

- * **Caso JUMP:** Viene eseguito il metodo `_processJump()` ricevendo in input il blocco corrente. Richiama il metodo `_getFlow()` pasandogli l'identificativo del flusso da eseguire attraverso il campo **target**. Il metodo citato richiede ad **Azzurra.io** il flusso che ha l'identificativo uguale a quello ricevuto in input, e una volta ricevuto comincia l'esecuzione del nuovo flusso conversazionale richiamando `start()`;
- * **Caso IF:** Viene eseguito il metodo `_processIf()` passando in input il blocco corrente. Valuta la condizione attraverso `_manageConditions()`, e in base al risultato stabilisce quale sarà il blocco di conversazione successivo da eseguire;
- * **Caso PROD:** Viene eseguito il metodo `_processProd()` passandogli in input il blocco corrente. Stabilisce che formattazione deve essere fatta e la esegue attraverso il metodo `_manageExpressions()`. Infine, passa all'esecuzione del prossimo blocco;
- * **CALLFUNC:** Viene ricavato il *payload* del blocco e codificato il *template* in handlebars per generare il corpo della richiesta, una volta fatto ciò la richiesta è pronta e viene mandata a **Azzurra.io**. Successivamente viene salvata la risposta sulla variabile indicata del campo **var** e in base alla risposta, se andata a buon fine oppure no, si eseguirà il corrispondente blocco successivo.

7. Nella Figura 4.11 viene mostrato il caso in cui viene eseguito il metodo `_processAsk()` perché il blocco attuale è di tipo **ASK**;
8. Viene eseguito `_renderText()` per costruire il testo della domanda;
9. Attraverso `_buildQuestions()` vengono creati gli **BlockItems** delle possibili scelte;
10. Viene eseguito il metodo `_processWidgetOptions()` per creare le eventuali **widgetOptions**;
11. A questo punto la struttura del messaggio è stata creata, manca solo la visualizzazione nella *chat* che è compito di **ChatComponent**, perciò attraverso il metodo `next()` di Angular, viene emesso un segnale che avvisa a chi è in ascolto su `_outgoingMessageSub$` che è stato creato un nuovo messaggio e che occorre visualizzarlo;
12. Viene notificato agli ascoltatori l'evento descritto al punto precedente;
13. Nel **AzzuraService** c'è un ascoltatore che riceve i segnali mandati dai metodi di **FlowService**, perciò con la notifica inviata al punto precedente **AzzuraService** ha il messaggio che è stato appena creato;
14. **AzzuraService** emettere a sua volta il nuovo messaggio appena ricevuto, verso l'ascoltatore presente in **ChatComponent**;

15. Quando **ChatComponent** riceve il messaggio eseguirà il metodo **createMessage()** di **ChatService** passandogli il messaggio ricevuto. Questo metodo si occuperà di far creare il messaggio grafico nel modo corretto. Innanzitutto, verifica chi ha emesso il messaggio, se è stato l'utente umano o il bot **Azzurra**. Nel caso in cui il messaggio sia del bot **Azzurra** viene indicato il testo da mostrare estraendolo dal messaggio ricevuto in input e aggiunto lo sprite di Azzurra per indicare graficamente che il messaggio viene dal bot;
16. Nel caso in cui il messaggio preveda delle azioni da parte dell'utente umano, ovvero ci siano dei **Widget**, viene chiamato il metodo **createAction()** passandogli sempre il messaggio. In questo metodo viene verificato che tipo di **Widget** deve essere creato, per ogni **Widget** c'è un corrispondente metodo che ne imposta il testo da far visualizzare quando l'utente interagisce con esso, tale testo potrebbe essere un testo di default o un testo indicato nel campo **widgetOptions** inoltre nel caso del **DATEPICKER** o **TIMEPICKER**, viene impostato il formato di visualizzazione del giorno e dell'ora;
17. Nel caso rappresentato dalla Figura 4.11, viene chiamato il metodo **_createQRcodeAction()** il quale imposta il testo da mostrare e chiama il metodo **_openQRcode()**;
18. **_openQRcode()** Richiama il metodo **create()** per creare il **Widget QRCODE**;
19. Con il metodo **create()** si crea il ModalController di Ionic per poter utilizzare la componente Ionic che gestisce il lettore di QR code. Si crea quindi graficamente il **Widget**;
20. Terminata la creazione il messaggio viene ritornato il messaggio costruito a **ChatComponent**;
21. Con **onNewMessage()** viene inserito nella *chat* il nuovo messaggio;
22. Il nuovo messaggio viene inviato a **Azzurra.io**, per essere memorizzato, attraverso il metodo **addNewMessageToCache()**;
23. Viene aggiornata la *chat* per mostrare il nuovo messaggio con il metodo **_updateMessageVisibility()**;
24. **_updateMessageVisibility()** chiama **_showLoading()** per simulare un caricamento;
25. Infine, **_updateMessageVisibility()** chiama **_scrollDown()** per muovere verso il basso la *chat* in modo da mostrare il nuovo messaggio che altrimenti rimarrebbe nascosto.

Una volta che l'utente interagisce con il **Widget** deve essere creato il messaggio dell'utente umano con la sua risposta.

4.3.2 Messaggio dell'utente umano

La generazione di un messaggio da parte dell'utente umano prevede le seguenti azioni:

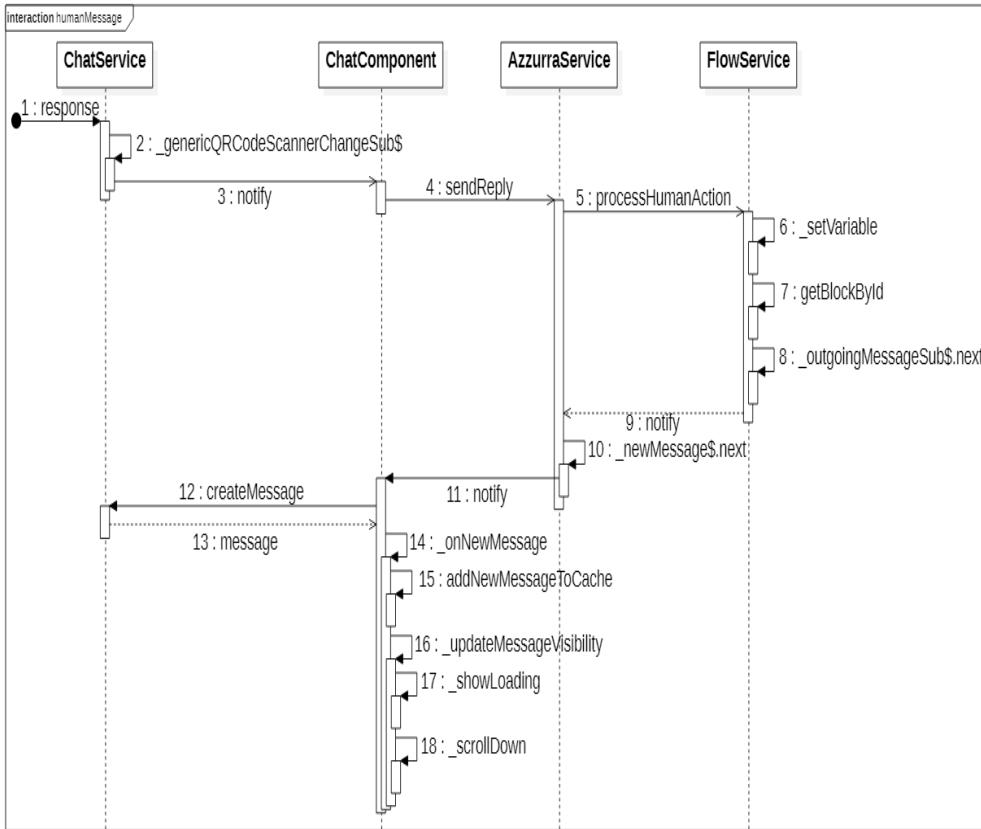


Figura 4.12: Diagramma di sequenza del processo di generazione del messaggio del bot Azzurra

1. Quando l'utente esegue un'azione che genera una risposta da parte sua, il **ChatService** emette il valore della risposta ogni volta che un **Widget** la riceve;
2. Nel **ChatComponent** esiste un ascoltatore per ogni evento emesso da ogni **Widget**. Nella Figura 4.12 viene rappresentato il caso in cui il **Widget** sia di tipo **QRCODE** perciò, attraverso il metodo `next()` di Angular, viene emesso un segnale che avvisa a chi è in ascolto su `_genericQRCodeScannerChangeSub$` che è stato emesso un evento da **QRCODE** inserendo anche il valore letto dallo scanner;
3. Viene notificato agli ascoltatori l'evento descritto al punto precedente;
4. **ChatComponent** riceve la notifica e chiama il metodo `sendReply()` di **AzzurraService** dandogli in input il valore letto dallo scanner ricevuto dalla notifica solo se il messaggio è valido, se non lo è viene ignorato;

5. Il metodo **sendReply()** chiama a sua volta **processHumanAction()** di **FlowService**;
6. In **processHumanAction()** viene creata la struttura del messaggio con la risposta dell'utente perciò, viene richiamato il metodo **_setVariable()** in cui, viene impostato il valore della risposta ritornato dal **Widget** che è stato salvato nella variabile indicata nel campo **var**;
7. In **processHumanAction()** viene richiamato il metodo **getBlockId()** che verifica se il **Widget** abbia al suo interno un proprio campo **target**, se sì viene impostato come prossimo blocco di conversazione il blocco che ha il codice identificativo uguale a quello contenuto nel **target**, se invece non ha un campo **target** proprio si va a prendere il valore del campo **target** del blocco e si imposta il suo valore come prossimo blocco da eseguire;
8. Attraverso il metodo **next()** di Angular, viene emesso un segnale che avvisa a chi e in ascolto su **_outgoingMessageSub\$** che è stato emesso un nuovo messaggio;
9. Viene notificato agli ascoltatori l'evento descritto al punto precedente;
10. **AzzurraService** riceve la notifica e tramite il metodo **next()** avvisa e invia la struttura del nuovo messaggio a tutti coloro che sono in ascolto su **_newMessage\$**;
11. Viene notificato agli ascoltatori l'evento descritto al punto precedente;
12. **ChatComponent** riceve il messaggio e chiama il metodo **createMessage()** di **ChatService** passandogli il messaggio ricevuto. Questo metodo si occuperà di far creare il messaggio grafico nel modo corretto. In questo caso il messaggio è di tipo umano perciò verrà creato secondo una certa specifica;
13. Viene ritornato il messaggio a **ChatComponent**;
14. Infine, per visualizzare il messaggio sulla *chat*, vengono rifatte le stesse operazioni che erano state fatte per il messaggio del bot **Azzurra**.

5 | FLUSSI CONVERSAZIONALI PRODOTTI

In questo capitolo verrà descritto il lavoro che è stato fatto di analisi, progettazione e implementazione dei flussi conversazionali per Azzurra creati durante lo stage.

5.1 Analisi dei requisiti

5.1.1 Descrizione del problema

Durante lo stage è stato deciso, di comune accordo con il tutor aziendale, di costruire due flussi conversazionali per il bot Azzurra, nello specifico:

- * **DeskBooking:** Questo flusso conversazionale consiste nel gestire le prenotazioni di un posto a sedere. Deve esserci la possibilità di richiedere una nuova prenotazione, visualizzare la lista delle proprie prenotazioni e infine, visto che è richiesto, che per riscattare il posto a sedere, quando si sta per iniziare a usufruire del posto si deve scannerizzare un QR code, messo nel posto a sedere, per poter verificare se chi ha scansionato il QR code ha veramente diritto a usufruire del posto, c'è perciò bisogno di integrare un lettore di QR code in Azzurra per poter fare il controllo. Perciò, si deve poter aprire la fotocamera, scannerizzare il QR code che verrà usato per controllare se il lavoratore può usufruire del posto dopo il controllo, comunicare l'esito della verifica al lavoratore;
- * **Planning:** Questo flusso conversazionale consiste nel far visualizzare al lavoratore il lavoro che deve svolgere. Deve esserci la possibilità di richiedere la visualizzazione del lavoro pianificato di uno specifico giorno oppure la possibilità di vedere il lavoro pianificato per tutta la settimana;

5.1.2 Requisiti

Ogni requisito sarà strutturato come segue:

- * Identificativo: **R[Importanza][Tipologia][Codice]**
Dove:

- **Importanza:**

- * **1:** Requisito obbligatorio, ovvero irrinunciabile per almeno uno degli stakeholder
 - * **2:** Requisito desiderabile, ovvero non strettamente necessario ma che porta valore aggiunto riconoscibile
 - * **3:** Requisito opzionale, ovvero relativamente utile oppure contrattabile più avanti nel progetto

- **Tipologia:**

- * **F:** Funzionale, definisce una funzione di un sistema di uno o più dei suoi componenti
 - * **Q:** Qualitativo, definisce un requisito per garantire la qualità per un certo aspetto del progetto
 - * **P:** Prestazionale, definisce un requisito che garantisce efficienza prestazionale nel prodotto
 - * **V:** Vincolo, definisce un requisito volto a far rispettare un dato vincolo
- **Codice:** Viene utilizzato per identificare univocamente il requisito tramite un numero progressivo

Dopo un'analisi del problema sono stati individuati i seguenti requisiti

Codice	Descrizione
R1F1	Il lavoratore deve poter accedere alla funzionalità di prenotazione posto.
R1F2	Il lavoratore deve poter inserire una nuova prenotazione di un posto a sedere.
R1F3	Il lavoratore deve poter visualizzare le sue prenotazioni.
R1F4	Il lavoratore deve poter scansionare il QR code per poter usufruire del posto prenotato.
R1F5	Il lavoratore deve poter inserire la data in cui vuole prenotare il posto a sedere se disponibile.
R1F6	Il lavoratore deve poter inserire l'ora di inizio della prenotazione che desidera.
R1F7	Il lavoratore deve poter inserire l'ora di terminazione della prenotazione che desidera.
R1F8	Il lavoratore deve poter inserire la stanza del posto a sedere che desidera prenotare.
R1F9	Il lavoratore deve poter inserire il posto a sedere che desidera prenotare se disponibile.
R1F10	Il lavoratore deve poter visualizzare il messaggio di conferma se la prenotazione del posto a sedere è andata a buon fine.
R1F11	Il lavoratore deve poter visualizzare il messaggio d'errore se non è stato possibile inserire la nuova prenotazione.
R1F12	Il lavoratore deve poter visualizzare le sue prenotazioni del giorno corrente.
R1F13	Il lavoratore deve poter visualizzare le sue prenotazioni del giorno successivo.
R1F14	Il lavoratore deve poter visualizzare le sue prenotazioni di uno specifico giorno.
R1F15	Il lavoratore deve poter inserire la data del giorno in cui vuole vedere le prenotazioni.
R1F15	Il lavoratore, per ogni prenotazione, deve poter visualizzare l'ora di inizio della prenotazione.
R1F17	Il lavoratore, per ogni prenotazione, deve poter visualizzare l'ora di terminazione della prenotazione.

Tabella 5.1: Tabella del tracciamento dei requisiti

Codice	Descrizione
R1F18	Il lavoratore, per ogni prenotazione, deve poter visualizzare la stanza della prenotazione.
R1F19	Il lavoratore, per ogni prenotazione, deve poter visualizzare il posto della prenotazione.
R1F20	Il lavoratore, dopo avere scannerizzato il QR code del posto a sedere, deve ricevere un messaggio di conferma se può usufruire del posto a sedere.
R1F21	Il lavoratore, dopo avere scannerizzato il QR code del posto a sedere, deve ricevere un messaggio d'errore se non può usufruire del posto a sedere.
R1F22	Il lavoratore deve poter visualizzazione la pianificazione di uno specifico giorno.
R1F23	Il lavoratore deve poter visualizzazione la pianificazione della settimana corrente.
R1F24	Il lavoratore deve poter inserire la data del giorno in cui vuole vederne la pianificazione del lavoro a lui assegnato.
R1F25	Il lavoratore deve poter visualizzare la data del giorno del lavoro pianificato.
R1F26	Il lavoratore deve poter visualizzare l'ora d'inizio del lavoro pianificato.
R1F27	Il lavoratore deve poter visualizzare l'ora di terminazione del lavoro pianificato.
R1F28	Il lavoratore deve poter visualizzare il lavoro che è stato pianificato per essere svolto.
R1V1	Per implementare i flussi conversazionali devono essere usati Angular e Ionic.
R1V2	Per gestire la fotocamera per la lettura del QR code deve essere usato il plugin di Cordova, QR Scanner.

Tabella 5.2: Tabella del tracciamento dei requisiti

5.2 Progettazione

Dopo aver individuato i requisiti che descrivono i flussi da costruire, si è passati alla progettazione dei flussi. Come spiegato nel capitolo precedente i flussi conversazionali non sono altro che un insieme di blocco che asseconda del tipo di appartenenza svolgono determinate funzioni. Grazie a ciò, la progettazione dei due flussi è iniziata con l'inserimento dei blocchi corretti e il collegamento tra di essi, ottenendo così due diagrammi che rappresentano i due flussi dove viene mostrato che passi deve fare il bot Azzurra durante la conversazione con l'utente umano. Successivamente si sono progettati i metodi da aggiungere a quelli esistenti per poter creare i messaggi nel modo corretto.

Di seguito vengono illustrati i diagrammi dei flussi fatti.

5.2.1 Gestione delle prenotazioni dei posti

Nei seguenti diagrammi viene mostrato l'insieme dei blocchi che fanno parte del flusso conversazionale **DeskBooking**. Per comodità il flusso invece di usare un unico diagramma molto grande si è deciso di rappresentare il flusso attraverso tre diagrammi più piccoli.

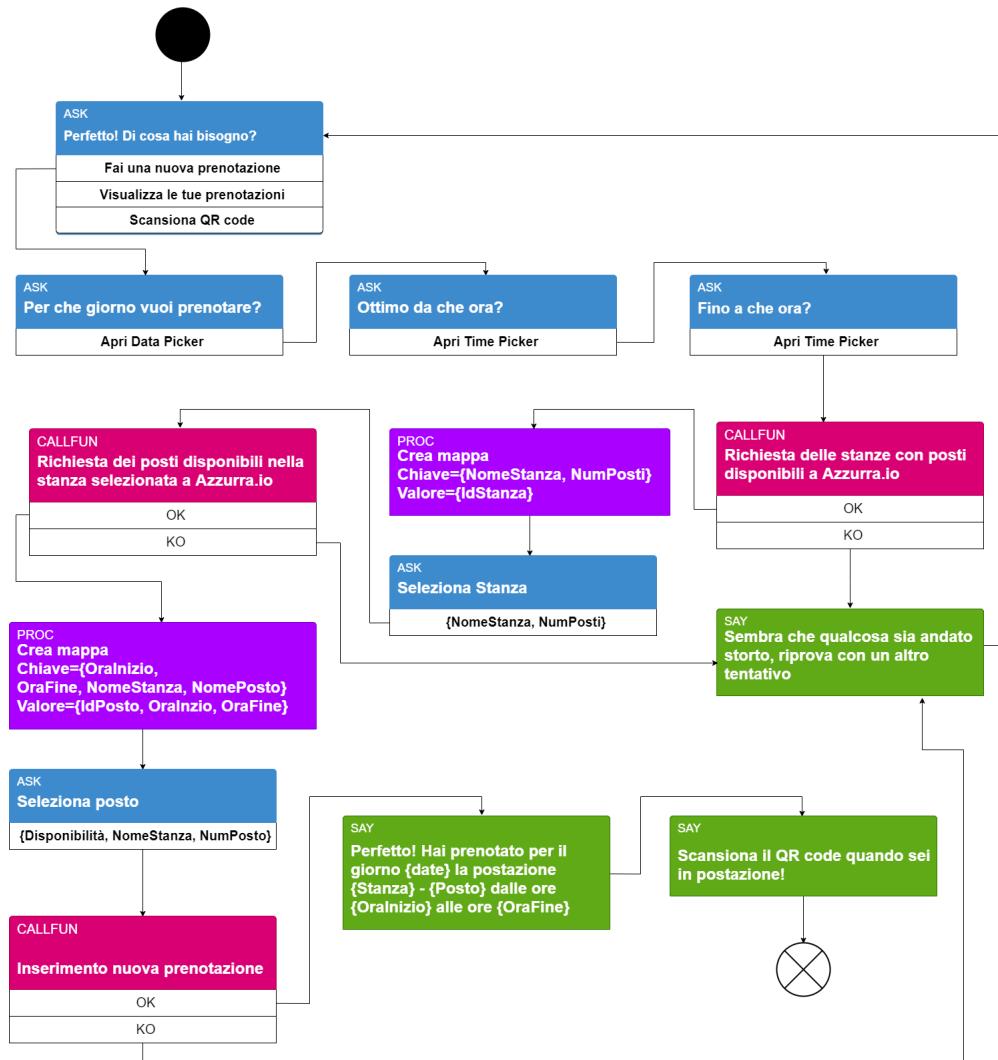


Figura 5.1: Diagramma per l'inserimento di una nuova prenotazione del flusso DeskBooking

La Figura 5.1 rappresenta il ramo del flusso **Deskbook** dedicato all'inserimento di una nuova prenotazione. È così composto:

1. Il flusso inizia con un blocco **ASK** che chiede all'utente se vuole inserire una nuova prenotazione o scansionare un QR code;
2. Nel caso in cui l'utente voglia inserire una nuova prenotazione, attraverso un blocco **ASK** viene chiesta la data che vuole inserire per la prenotazione. Viene progettato che l'inserimento della data viene fatta attraverso il **DATEPICKER**;
3. Successivamente tramite un blocco **ASK** viene chiesta l'ora di inizio per la prenotazione. L'inserimento dell'ora avviene tramite il **TIMEPICKER**;
4. Viene rifatta la stessa operazione del punto precedente ma chiedendo la data di terminazione della prenotazione;

5. Terminato il punto precedente, l'utente ha inserito l'intervallo di tempo all'interno del quale desidera effettuare una prenotazione di un posto a sedere. Attraverso il blocco **CALLFUN** viene chiesto a **Azzurra.io** quali stanze con posti liberi sono disponibili per la data e l'intervallo inseriti dall'utente;
6. Se non ci sono stanze con posti liberi o l'operazione di richiesta va in errore, attraverso un blocco **SAY** viene informato l'utente della situazione e ricomincia l'esecuzione dall'inizio del flusso;
7. Se invece ci sono stanze con posti liberi, attraverso il blocco **PROC** vengono formattati i dati ricevuti in modo da poterli mostrare in una forma adatta alla situazione. In questo caso viene chiesto di creare una mappa con chiave contenente il nome della stanza e il numero dei posti, mentre come valore l'identificativo della stanza. Quando si vorrà mostrare questi dati, verrà solo mostrato la chiave dei dati;
8. Tramite il blocco **ASK** vengono mostrate le stanze disponibili mostrando i dati secondo la formattazione fatta al punto precedente;
9. L'utente sceglie la stanza è viene controllato se nel frattempo è ancora disponibile e chiede quali posti a sedere sono liberi;
10. Se avviene un errore di connessione o non ci sono più posti liberi si torna al punto 6 spiegato precedentemente;
11. I dati ricevuti vengono formattati attraverso il blocco **PROC** creando la mappa con chiave contenente ora di inizio, orario di terminazione, nome stanza e nome posto a sedere, mentre come valore conterrà l'identificativo del posto a sedere, l'orario di inizio e di fine;
12. Viene chiesto all'utente, attraverso un blocco **ASK**, di scegliere uno dei posti a sedere liberi;
13. Viene contattato **Azzurra.io** con il blocco **CALLFUN** per inserire la nuova prenotazione;
14. Se avviene un errore nell'inserimento viene eseguito il punto 6;
15. Se l'operazione va a buon fine viene comunicato all'utente l'esito positivo dell'operazione, ricordandogli i dati della prenotazione e di scansionare il QR code per riscattare il posto a sedere. Il flusso poi termina.

La Figura 5.2 rappresenta il ramo del flusso **Deskbook** dedicato alla visualizzazione delle prenotazioni. È così composto:

1. Il flusso inizia con un blocco **ASK** che chiede all'utente se vuole inserire una nuova prenotazione o scansionare un QR code;
2. Nel caso in cui l'utente voglia visualizzare le sue prenotazioni, viene chiesto attraverso un blocco **ASK** se vuole sapere le prenotazioni del giorno corrente o del giorno successivo o di un altro giorno;
3. Nel caso l'utente voglia vedere le sue prenotazioni del giorno corrente o del giorno successivo verrà fatta una richiesta a **Azzurra.io** per ottenere le prenotazioni della data inserita dall'utente. La richiesta viene fatta attraverso il blocco **CALLFUN**;

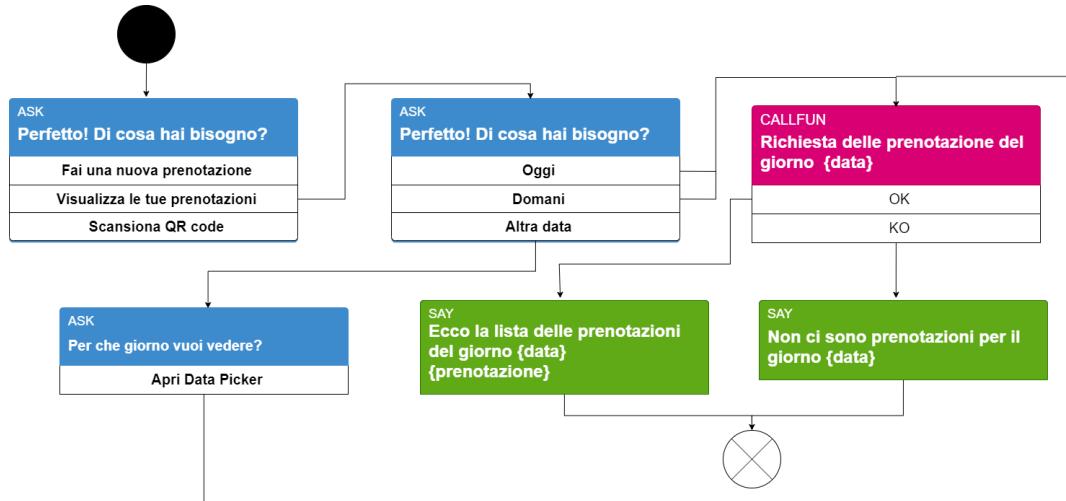


Figura 5.2: Diagramma per la visualizzazione delle prenotazioni del flusso DeskBooking

4. Se invece l'utente vuole vedere le sue prenotazioni di una data diversa dal giorno corrente o successivo, attraverso un blocco **ASK** viene chiesta la data che vuole inserire per la visualizzazione. L'inserimento della data viene fatta attraverso il **DATEPICKER**, successivamente si esegue il punto 3 per la richiesta;
5. Se ci sono prenotazioni queste vengono mostrate all'utente, se invece avviene un errore o non ci sono prenotazioni fatte da lui, verrà avvisato di tale evento. Dopo questo passo il flusso termina.

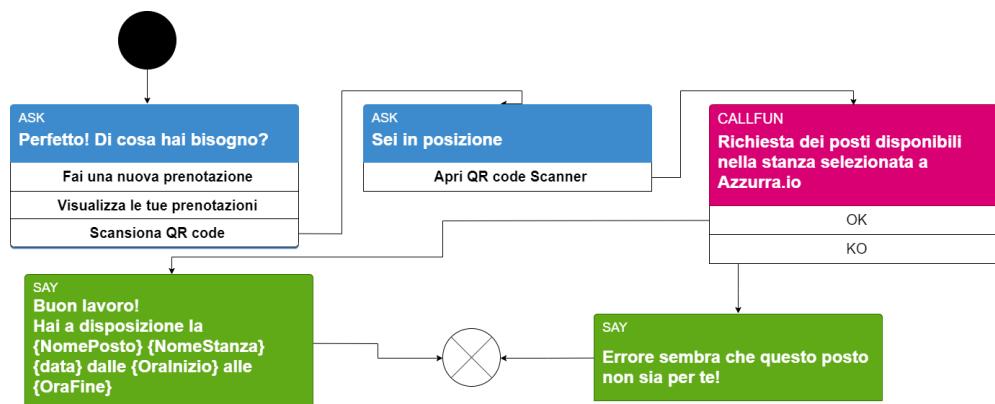


Figura 5.3: Diagramma per lo scansionamento del QR code del flusso DeskBooking

La Figura 5.3 rappresenta il ramo del flusso **Deskbook** dedicato allo scansionamento del QR code. È così composto:

1. Il flusso inizia con un blocco **ASK** che chiede all'utente se vuole inserire una nuova prenotazione o scansionare un QR code;

2. Nel caso in cui l'utente voglia scansionare un QR code per riscattare il suo posto a sedere prenotato, viene chiesto attraverso un blocco **ASK** di aprire il scannerizzatore di QR code. Viene usato **QRSCANNER** per leggere il QR code;
3. Viene chiesto a **Azzurra.io** attraverso il blocco **CALLFUN**, se il posto a sedere può essere usato dall'utente;
4. Se l'esito è positivo, viene comunicato all'utente che può usufruire del posto fino al termine della prenotazione. Termina così il flusso;
5. Se l'esito è negativo, viene informato l'utente che non può usare il posto a sedere in quel momento. Termina così il flusso;

5.2.2 Visualizzazione della pianificazione

Nel seguente diagramma viene mostrato l'insieme dei blocchi che fanno parte del flusso conversazionale **Planning**.

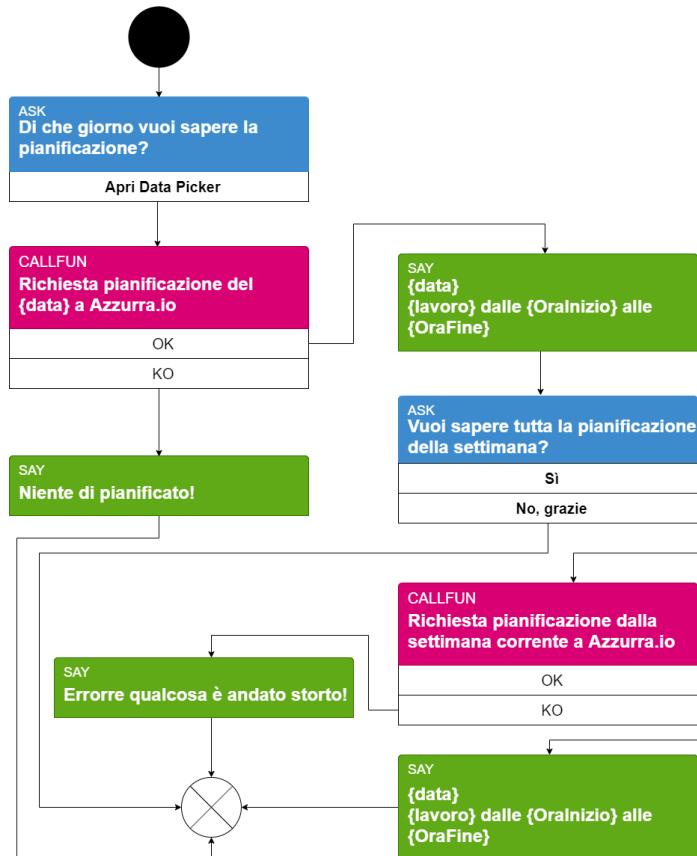


Figura 5.4: Diagramma per l'inserimento di una nuova prenotazione del flusso DeskBooking

La Figura 5.4 rappresenta il ramo del flusso **Planning** dedicato alla visione della pianificazione del lavoro da svolgere.

1. Il flusso inizia con un blocco **ASK** che chiede all'utente di quale giorno vuole vedere la pianificazione. L'inserimento della data viene fatta attraverso il **DATEPICKER**;
2. Viene fatta richiesta a **Azzurra.io**, utilizzando il blocco **CALLFUN**, di trovare la pianificazione del giorno indicato dall'utente;
3. Se non viene trovata nulla allora l'utente viene avvisato tramite un blocco **SAY** che non c'è niente di pianificato e il flusso termina;
4. Se invece c'è una pianificazione disponibile per il giorno indicato dall'utente, viene visualizzata attraverso un blocco **SAY**;
5. Dopo il punto precedentemente descritto viene chiesto con un blocco **SAY** se si vuole sapere la pianificazione di tutta la settimana;
6. Se l'utente risponde no il flusso termina;
7. Se l'utente risponde sì viene fatta richiesta a **Azzurra.io**, utilizzando il blocco **CALLFUN**, di trovare la pianificazione della settimana corrente;
8. Se la richiesta va buon fine viene mostrata la pianificazione della settimana, altrimenti viene mostrato un messaggio. In entrambi i casi il flusso poi termina.

5.3 Codifica

Per implementare i due flussi si sono utilizzati i *framework* Angular e Ionic. Grazie a Angular si è potuto strutturare un'applicazione web come una gerarchia di componenti quindi, attraverso il linguaggio TypeScript si è gestita l'*application logic* mentre con HTML e CSS si è gestita la *presentation logic*. Purtroppo solo l'uso di Angular non basta per poter sviluppare un'applicazione *mobile* e non web, si è quindi usato Cordova, un *framework* che permette di sviluppare un'applicazione mobile multi-piattaforma, quindi sia per Android e sia per iOS, con tecnologie web e inoltre, offre API per accedere alle funzionalità native del dispositivo, ad esempio la fotocamera. Infatti, Cordova incapsula l'applicazione web e la esegue localmente all'interno di un'applicazione nativa che può interagire con le funzionalità del dispositivo. Per sfruttare le funzionalità di Angular e di Cordova assieme, è stato usato il *framework* Ionic che permette di creare un ambiente integrato che semplifica lo sviluppo di applicazioni offrendo inoltre, componenti grafiche ottimizzate per i dispositivi *mobile*.

Per quanto riguarda la codifica, per prima cosa si è implementato una configurazione JSON per ogni flusso, dove si sono codificati i vari blocchi progettati, utilizzando la sintassi spiegata nel precedente capitolo. Una volta scritte le due configurazioni si è dovuto aggiornare il *main flow* aggiungendo nel primo blocco che viene eseguito, cioè un blocco **ASK** dove viene chiesto che funzionalità si vuole eseguire, due **BlockItem** per indicare le due nuove funzionalità offerte dai due flussi prodotti. Oltre alle due nuove scelte, nel *main flow* sono stati aggiunti due blocchi **JUMP** per permettere di mandare in esecuzione i due nuovi flussi quando l'utente ne richiede l'esecuzione subito dopo la selezione della funzionalità desiderata da parte dell'utente.

Successivamente era necessario creare i metodi per gestire le componenti grafiche dei **WIDGET**, in questo caso **DATEPICKER**, **TIMEPICKER** e **QRSCANNER**.

Quindi, nel metodo `createActions()` di **ChatService**, utilizzato per la gestione dei **WIDGET**, si sono aggiunti i metodi per impostare il testo che devono mostrare i tre **WIDGET** citati prima. Inoltre, ognuno dei tre metodi chiamano i metodi dedicati per la creazione dei **WIDGET**, anche questi metodi non esistevano e sono stati creati, in particolare per il metodo che crea il **QRSCANNER**, `_openQRcode()`, viene utilizzato il ModalController di Ionic per creare la classe dove è definita l'interfaccia grafica e i metodi per il funzionamento di **QRSCANNER**. Il ModalController di Ionic permette di aprire una nuova finestra sopra a quella corrente per visualizzare la componente Ionic definita nella nuova finestra, in questo caso la classe che implementa il lettore di QR code. Una volta finito di usare la nuova finestra può essere chiusa e ritornare alla finestra precedente che sarà nello stato in cui era prima dell'apertura della nuova finestra. Riguardo invece alla classe che gestisce il lettore QR code, denominata CameraComponent, al suo interno richiama il plugin di Cordova, QR Scanner. QR Scanner è un API che permette di accedere alla fotocamera del dispositivo e di scansionare i QR code. Vengono perciò definiti due metodi in **CameraComponent**, un metodo per l'apertura della fotocamera e la lettura del QR code e per la chiusura della fotocamera che dopo la chiusura, invia l'eventuale valore letto al ModalController. Nel **CameraComponent** viene definito anche il suo aspetto grafico mostrato nella Figura 4.9.

Nel **ChatComponent** invece viene implementato il metodo `_initGenericQRCode()` il quale aspetta di ricevere il valore letto dal lettore di QR code, che una volta che lo riceve, richiama il metodo `sendReply()` di **AzzurraService** per dare inizio al processo di creazione del messaggio dell'utente umano spiegato nel capitolo precedente.

Per quanto riguarda i metodi per il **DATEPICKER** e per il **TIMEPICKER**, essi sono molto simili a quelli per gestire il lettore QR code, l'unica differenza è che i metodi analogi a `_openQRcode()` quindi `_openDatePicker()` e `_openTimePicker()` non utilizzano il ModalComponent ma si utilizzano l'ion-datetime, una componente grafico offerto da Ionic che può essere configurata per chiedere una data oppure un intervallo temporale, il risultato viene mostrato nella Figura 4.8 e nella Figura 4.7. Quindi in questi due metodi viene definito come si devono presentare i due *picker* e quindi non c'è nemmeno bisogno di un component apposito che li gestisca come per **QRSCANNER**.

5.4 Risultati

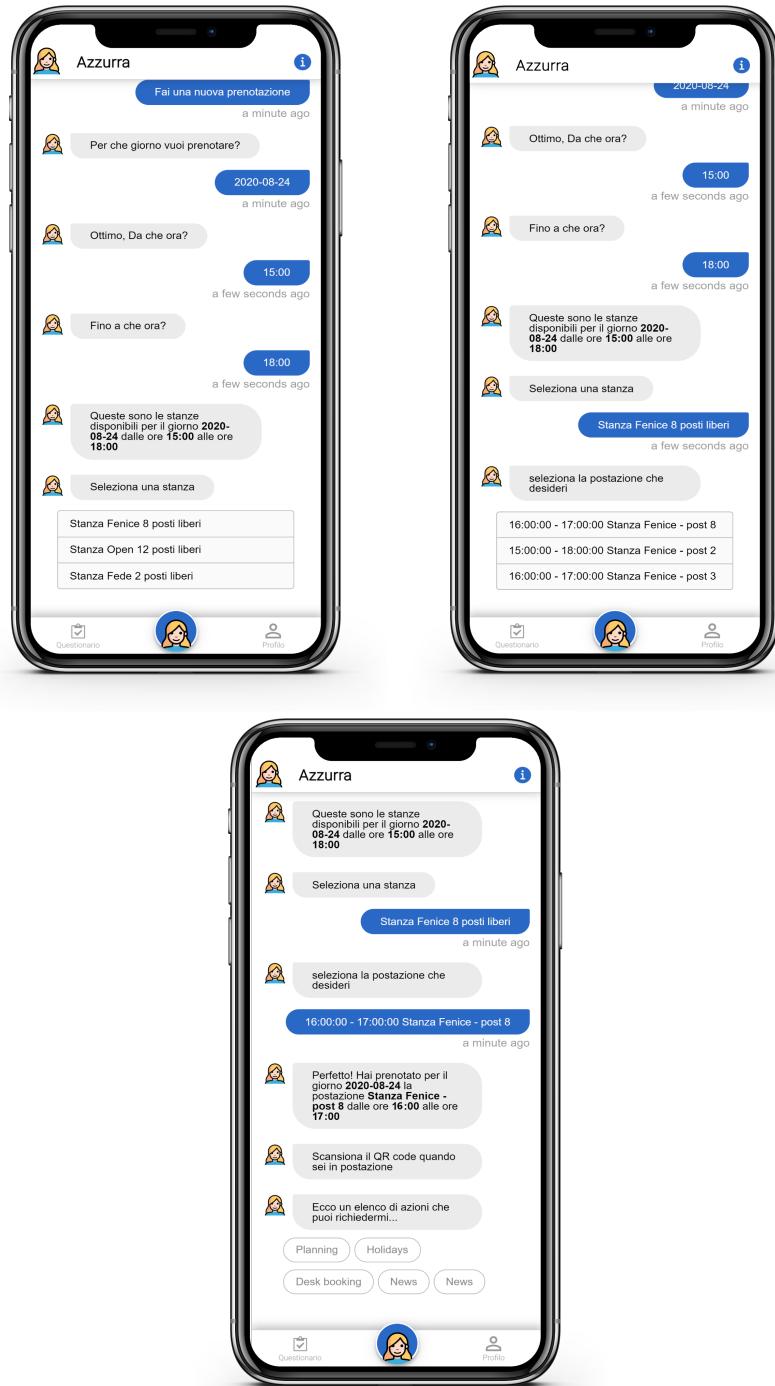


Figura 5.5: Prenotazione



Figura 5.6: Prenotazione



Figura 5.7: Prenotazione



Figura 5.8: Prenotazione

6 | VERIFICA E VALIDAZIONE

7 | CONCLUSIONI

- 7.1 Consuntivo finale**
- 7.2 Raggiungimento degli obiettivi**
- 7.3 Conoscenze acquisite**
- 7.4 Valutazione personale**

A | APPENDICE A

Citazione

Autore della citazione

ACRONIMI E ABBREVIAZIONI

API Application Program Interface. 14, 23, 41

AWMS Advanced Workforce Management System. 1–5, 41

E2E Test End to End. 5, 6, 9

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. 43

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.. 43

Bot È un software progettato per simulare una conversazione con un essere umano. Lo scopo principale di questi software è quello di simulare un comportamento umano e sono a volte definiti anche agenti intelligenti e vengono usati per vari scopi come la guida in linea, per rispondere alle FAQ degli utenti che accedono a un sito. In alcuni utilizzano sofisticati sistemi di elaborazione del linguaggio naturale, ma molti si limitano a eseguire la scansione delle parole chiave nella finestra di input e fornire una risposta con le parole chiave più corrispondenti. 3, 5, 6

Electrolux Electrolux è una multinazionale svedese produttrice di elettrodomestici con sede a Stoccolma. 1

Machine learning Nell'ambito dell'informatica, l'apprendimento automatico è una variante alla programmazione tradizionale nella quale in una macchina si pre-dispone l'abilità di apprendere qualcosa dai dati in maniera autonoma, senza istruzioni esplicite. 2

Plant manager Detto anche responsabile di stabilimento, è colui che presiede e organizza le operazioni quotidiane degli impianti di produzione aziendali, di cui deve assicurare il funzionamento ottimale ed efficiente. Si occupa dei lavoratori, assegnando funzioni e ruoli, definendo orari di lavoro e produzione, formando i neo assunti. Raccoglie e analizza i dati di produzione per trovare eventuali spazi di miglioramento. Si occupa della sicurezza dei lavoratori e quella degli impianti; monitora le apparecchiature di produzione e, in caso di necessità, della loro riparazione o sostituzione. 2, 3

QR-CODE È un codice a barre bidimensionale (o codice 2D), ossia a matrice, composto da moduli neri disposti all'interno di uno schema bianco di forma quadrata,

impiegato tipicamente per memorizzare informazioni generalmente destinate a essere lette tramite uno smartphone. 3

SCRUM È un framework agile per la gestione del ciclo di sviluppo del software, iterativo ed incrementale, concepito per gestire progetti e prodotti software o applicazioni di sviluppo. Nel proprio manifesto prevede i seguenti punti che lo caratterizzano, le persone e le interazioni sono più importanti dei processi e dei strumenti, meglio avere da subito software funzionante che documentazione ampia, meglio una collaborazione con il cliente piuttosto che fare una negoziazione del contratto, essere in grado di rispondere ai cambiamenti piuttosto che rispettare un piano. I progetti Scrum progrediscono attraverso una serie di sprint che hanno una durata massima di un mese. Negli sprint vengono decisi quali requisiti devono essere soddisfatti, e quindi successivamente, progettati, implementati e testati. 7, 12

E2E Con il test end-to-end (end-to-end testing) si intende quell'attività di testing dell'interfaccia grafica come la vedono gli utenti del programma dall'inizio alla fine. In altre parole rappresenta una metodologia utilizzata per verificare se il flusso di un'applicazione si sta comportando come progettato dall'inizio alla fine senza che vengano rilevati dei failure che andrebbero a inficiare sulla qualità dell'applicazione stessa. 43

WebSocket È una tecnologia web che fornisce canali di comunicazione a due direzioni cioè gli interlocutori possono sia inviare sia ricevere contemporaneamente attraverso una singola connessione TCP. 3, 5

BIBLIOGRAFIA
