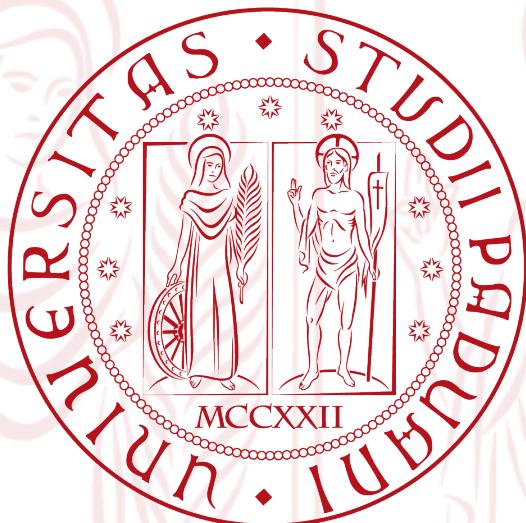


Università degli Studi di Padova

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 trecentoquattro ore, dal laureando Federico Perin presso l'azienda AzzurroDigitale S.r.l. Lo scopo dello stage è essere introdotto all'interno del progetto aziendale "Azzura.flow". Tale progetto prevede lo sviluppo di un bot^[g] 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 informare il lavoratore sul suo piano di lavoro.

La prima attività svolta è 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* Udemy.

In secondo luogo è richiesto lo studio del funzionamento dell'architettura^[g] del sistema che permette l'esecuzione di Azzurra, in particolare il funzionamento dei metodi del motore conversazionale denominato Azzura Flow Engine. Successivamente all'apprendimento delle conoscenze previste, ho eseguito la progettazione e l'implementazione di alcuni flussi di conversazione per Azzurra. Inoltre è richiesto lo studio di un *template-engine* per permettere il supporto multi-lingua all'interno di Azzurra e successivamente la sua implementazione.

Infine, da buona prassi, si è svolta un'attività di documentazione affiancata a tutto il progetto, sia riguardante il codice ma anche le scelte progettuali e lo sviluppo di una *test-suite* di test End to End (E2E) per l'applicazione *mobile* e per il front-end^[g] in modo da verificare 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 ed il sostegno fornитоми durante tutto il lavoro.

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

Voglio inoltre ringraziare i miei amici per questi tre bellissimi anni trascorsi assieme e per avermi sempre sostenuto anche nei momenti più difficili.

Infine, desidero ringraziare i miei ex compagni di gruppo del progetto didattico del corso di Ingegneria del Software per aver reso più allegate le 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 Strumenti e tecnologie utilizzate	9
2.6.1 Tecnologie	9
2.6.2 Strumenti	11
2.7 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 AWMS	16
3.2 Operazioni	20
3.2.1 Creazione di una connessione attraverso WebSocket	20
3.2.2 Recupero di un flusso conversazionale	21
3.2.3 Richiesta e invio di dati	22
3.2.4 Gestione notifiche push	23
4 Azzurra Flow Engine	25
4.1 Cos'è	25
4.2 Flussi conversazionali	25
4.2.1 Shortcuts "shortcuts"	26
4.2.2 Configurazione "config"	26
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
4.4	Gestione dell'internazionalizzazione di AWMS	40
5	Flussi conversazionali prodotti	43
5.1	Analisi dei requisiti	43
5.1.1	Descrizione del problema	43
5.1.2	Requisiti	43
5.2	Progettazione	46
5.2.1	Gestione delle prenotazioni dei posti	46
5.2.2	Visualizzazione della pianificazione	49
5.3	Codifica	51
5.4	Risultati	52
5.5	Considerazioni	54
6	Testing	55
6.1	Test End to End	55
6.2	Tecnologie per il testing	56
6.3	Convezione	59
6.4	Test eseguiti	60
6.5	Considerazioni	63
7	Conclusioni	65
7.1	Consuntivo finale	65
7.2	Raggiungimento degli obiettivi	66
7.2.1	Riepilogo	67
7.3	Consegna dei prodotti	67
7.3.1	Riepilogo	67
7.4	Analisi retrospettiva	68
7.4.1	Conoscenze acquisite	68
7.4.2	Competenze acquisite	69
7.4.3	Tecnologie e strumenti utilizzati	69
7.4.4	Valutazione personale	69
	Acronimi e abbreviazioni	71
	Glossario	73
	Bibliografia	79

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 e Chat con Azzurra	18
3.7	Sequence diagram per la creazione di una connessione attraverso WebSocket	20
3.8	Sequence diagram per il recupero di un flusso conversazionale	21
3.9	Sequence diagram per il recupero dei dati sul lavoratore	22
3.10	Sequence diagram per l'invio di notifiche push	23
4.1	Menu contenente le shortcuts disponibili	26
4.2	Esempio di messaggio prodotto da un blocco di tipo ASK	28
4.3	Esempio di messaggio prodotto da un blocco di tipo SAY	29
4.4	Rappresentazione grafica dei buttons	32
4.5	Rappresentazione grafica degli items	32
4.6	Rappresentazione grafica del picker	32
4.7	Rappresentazione grafica del date picker	33
4.8	Rappresentazione grafica del QR scanner	33
4.9	Rappresentazione grafica del BlockItem	34
4.10	Diagramma di sequenza per la generazione di un messaggio di Azzurra	35
4.11	Diagramma di sequenza per la generazione di un messaggio dell'utente	39
5.1	Diagramma per l'inserimento di una nuova prenotazione del flusso DeskBooking	47
5.2	Diagramma per la visualizzazione delle prenotazioni del flusso DeskBooking	48
5.3	Diagramma per lo scansionamento del QR code ^[g] del flusso DeskBooking	49
5.4	Diagramma per la visualizzazione della pianificazione del flusso Planning	50
5.5	Richiesta di visualizzazione della pianificazione	52
5.6	Richiesta di visualizzazione delle prenotazioni e scannerizzazione di un QR code	53
5.7	Richiesta di inserimento di una nuova prenotazione	53
6.1	Piramide dei test	56
6.2	Una parte del report dei test per il mobile generato da Cucumber . . .	58
6.3	Schermata del server Appium	59

ELENCO DELLE TABELLE

2.1	Tabella riassuntiva delle attività pianificate per il progetto di stage	9
5.1	Tabella del tracciamento dei requisiti	44
5.2	Tabella del tracciamento dei requisiti	45
6.1	Tabella del tracciamento dei test E2E	60
6.2	Tabella del tracciamento dei test E2E	61
6.3	Tabella del tracciamento dei test E2E	62
7.1	Tabella riassuntiva del consultivo delle attività per il progetto di stage	65
7.2	Tabella riassuntiva del consultivo delle attività per il progetto di stage	66
7.3	Tabella riassuntiva degli obiettivi pianificati del progetto di stage	67
7.4	Tabella riassuntiva dei prodotti pianificati del progetto di stage	68

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 documento;
- * per i 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 Pasquale, Jacopo Pertile e Antonio Fornari) fondarono la *startup*, puntando fortemente nelle nuove emergenti tecnologie che il mercato offriva. Il primo cliente fu l'azienda Electrolux^[g] con la quale, grazie a una forte attività collaborativa, fu sviluppata una piattaforma per la gestione della forza lavoro denominata Advanced Workforce Management System (AWMS) che tutt'ora continua a ricevere miglioramenti. Dopo il successo ottenuto dalla collaborazione con Electrolux^[g], i fondatori capirono che il mercato delle aziende manifatturiere è la nicchia sulla quale potevano 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 ed implementare i loro processi grazie alle tecnologie, non intese come sostitutive all'uomo, ma come mezzi che abilitano le persone a lavorare nel miglior modo possibile massimizzando lo sforzo lavorativo.

1.3 L'idea

1.3.1 Il contesto applicativo

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

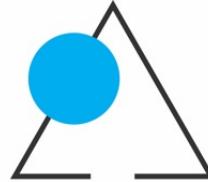


Figura 1.2: Logo di AWMS

AWMS è una soluzione *software* che utilizza algoritmi di machine learning^[g] per risolvere uno dei problemi cardine di un plant manager^[s] ovvero la pianificazione ottimale della forza lavoro che ha disposizione. L'obiettivo principale di tale soluzione è stabilire la persona giusta al posto giusto, in base alle competenze tecniche possedute del lavoratore, attraverso una pianificazione. Per permettere questo funzionamento, la piattaforma estrae dati sui lavoratori da database^[g] interni all'azienda che ha acquistato la soluzione, che ne descrivono le competenze possedute. Viene perciò registrato uno storico delle mansioni svolte per ogni lavoratore aggiornandolo nel tempo. Perciò, in base ai dati estratti dalla piattaforma, viene scelto il miglior candidato per un determinato compito.

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

1.3.2 Il progetto Azzurra.flow

Il progetto Azzurra.flow nasce dalla esigenza, da parte dell'azienda AzzurroDigitale, di offrire un prodotto completo per tutti i soggetti coinvolti nelle attività lavorative. Con AWMS si ha uno strumento che supporta i *team leader* o i *plant manager*^[g] 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 le seguenti funzionalità utili all'utente:

- * Visualizzare il proprio turno di lavoro;
- * Visualizzare i propri permessi lavorativi o richiederne di nuovi;
- * Visualizzare avvisi da parte dell'azienda;
- * Sapere qual'è il menu del giorno della mensa aziendale;
- * Poder effettuare prenotazioni di un posto a sedere in una sala riunioni e visualizzare le proprie prenotazioni utilizzando 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 denominato Azzurra Flow Engine, in grado di poter generare una conversazione con il lavoratore, attraverso l'interpretazione dei flussi di conversazione, anche essi da sviluppare, che indicano quali azioni deve fare Azzurra. Questi flussi devono essere memorizzati in un preciso posto e a questo proposito è stato progettato che sia un database^[g] contenuto nella nuova componente Azzurra.io con il compito non solo di tenere memorizzati i flussi conversazionali esistenti e di inviarli a Azzurra quando li richiede, ma anche di fare da tramite tra l'applicazione con all'interno Azzurra e AWMS, tutto attraverso una comunicazione tramite WebSocket^[g].

1.4 Organizzazione del testo

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

In seguito il documento sarà organizzato con la seguente struttura:

Il secondo capitolo descrive in modo dettagliato lo stage svolto, indicandone obiettivi, prodotti attesi, pianificazione delle attività, strumenti e tecnologie utilizzate e motivazioni personali.

Il terzo capitolo illustra l'architettura^[g] del sistema AWMS che permette il funzionamento di Azzurra. Vengono qui descritte le componenti dell'architettura e le varie operazioni tra essi.

Il quarto capitolo approfondisce il funzionamento del motore conversazionale di Azzurra indicando come avviene una conversazione tra Azzurra e l'utente.

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

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

Il settimo capitolo rappresenta la conclusione del documento in cui è proposto un riepilogo del lavoro svolto durante lo stage, degli obiettivi raggiunti ed 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 delle attività, i prodotti attesi, gli strumenti e tecnologie utilizzate durante lo stage, ed infine le motivazioni per cui ho scelto questo stage.

2.1 Descrizione dello stage

Lo stage è legato ad un progetto interno dell'azienda denominato "Azzurra.flow". Tale progetto nacque dall'esigenza dell'azienda AzzurroDigitale di offrire un prodotto più completo ai propri clienti che andasse ad affiancare la piattaforma AWMS. Perciò venne deciso di implementare un'applicazione *mobile* che potesse comunicare con la piattaforma AWMS, dando un mezzo di supporto al lavoratore di una azienda manifatturiera. All'interno di essa doveva essere implementato una *chat bot* con un bot^[g] denominato "Azzurra" che offrisse funzionalità di supporto al lavoratore. All'interno del progetto furono anche previste le implementazioni necessarie per la comunicazione tra AWMS e l'applicazione *mobile*, la gestione di una connessione attraverso WebSocket^[g] 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. Inoltre Azzurra.io ha il compito di fare da tramite tra l'applicazione *mobile* e AWMS.

Partendo dal progetto "Azzurra.flow" è stata ideata la proposta di stage a me rivolta, composta da attività che andassero a contribuire allo sviluppo del progetto.

Lo stage è stato costruito nelle seguenti parti:

- * Nella prima parte è 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 è supportato da video lezioni della piattaforma di *e-learning* Udeny, offerte dall'azienda;
- * La seconda parte è stata dedicata allo studio del funzionamento dell'architettura^[g] del sistema che permette l'esecuzione di Azzurra, in particolare il funzionamento dei metodi del motore conversazionale denominato Azzurra Flow Engine e in aggiunta, come esercitazione, è stato richiesta la creazione di alcuni *test E2E*^[g] per la parte front-end^[g] del sistema e per la *dashboard* di AWMS;
- * La terza parte è stata dedicata all'analisi, progettazione e implementazione di alcuni flussi di conversazione per il bot^[g] Azzurra;
- * La quarta parte è stata dedicata alla stesura della documentazione per la *Solution Design* di Azzurra;
- * Nella quinta parte, sulle basi dello studio fatto in precedenza, è stato previsto lo sviluppo di una test-suite di *test E2E*^[g], con l'obiettivo di automatizzare i test qualora le funzionalità dell'applicazione *mobile* funzionassero in modo corretto;

- * Infine, la sesta parte ed ultima parte è stata dedicata allo studio di alcuni aspetti dell'applicazione *mobile* che sono:
 - Gestione delle notifiche push^[g];
 - *Template* engine multi-lingua;
 - Gestione comportamenti *mobile application* in condizioni di mancanza di connettività.

2.2 Obiettivi

2.2.1 Classificazione

Viene ora riportato il piano di lavoro per il progetto di stage dell'anno accademico 2019/2020 svolto presso l'azienda AzzurroDigitale. 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

Era previsto lo svolgimento dei seguenti obiettivi:

Obbligatori

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

Desiderabili

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

2.3 Prodotti attesi

Durante lo stage era atteso lo sviluppo dei seguenti *deliverable*:

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

2.4 Modalità di svolgimento del lavoro

Lo stage è stato svolto in presenza negli uffici di AzzurroDigitale rispettando tutte le norme sul distanziamento sociale. L'orario di lavoro è stato dalle 9:00 fino alle 13:00 e dalle 14:00 fino alle 18:00. Durante lo stage sono stato inserito in un gruppo di sviluppatori che fornivano un'azione di supporto e guida per qualsiasi problema o difficoltà riscontrata durante le attività di stage. Nonostante ciò sono stato seguito anche dal mio tutor aziendale, *team leader* del gruppo di sviluppatori, che mi assegnava i *task* che dovevo realizzare e fissava gli obiettivi attesi per ciascuno di essi.

Durante lo stage per gestire le attività di progetto è stato utilizzato il modello agile SCRUM^[g], già adottato dall'azienda. Sono state quindi fissate le seguenti attività:

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

Infine, durante lo stage è stato mio compito redirigere un registro su cui, quotidianamente, segnare le attività svolte.

2.5 Pianificazione del lavoro

Per ognuna delle attività pianificate per i mesi di Luglio, Agosto e Settembre e in seguito illustrate in dettaglio, è stato chiesto di:

- * 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

In seguito viene riportata la pianificazione completa, basata su trecentoventi 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 application*).

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

- * **Formazione NestJS:** corso Udemy + *review* di alcuni componenti di "Azzurra" già sviluppati;
- * **End-to-end testing:** (Selenium + Protractor + Cucumber) lato front-end^[g];
- * **End-to-end testing:** (Appium + Protractor + Cucumber) lato *mobile application*.

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

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

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

- * Contributi alla redazione della *Solution Design* di “Azzurra”;
- * Contributi alla documentazione sorgenti di “Azzurra” (front-end^[g]/back-end^[g]).

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

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

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

- * Contributi alla redazione della *Solution Design* di “Azzurra”;
- * Implementazione Push Notifications^[g] (lato *mobile application*);
- * Implementazione Push Notifications^[g] (lato back-end^[g]).

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 application* 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	Date (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 dell'architettura di sistema di Azzurra, creazione di <i>test</i> per la <i>dashboard</i> di AWMS e per l'applicazione <i>mobile</i> .
40	13/07/2020 - 17/07/2020	Continuazione studio delle componenti del sistema di Azzurra, analisi, progettazione e implementazione di flussi conversazionali.
40	20/07/2020 - 24/07/2020	Documentazione per le componenti di Azzurra.
40	27/07/2020 - 31/07/2020	Continuazione studio di altre componenti di AWMS.
40	03/08/2020 - 07/08/2020	Documentazione delle componenti AWMS e implementazione notifiche push ^[g] .
40	17/08/2020 - 21/08/2020	Progettazione, implementazione e documentazione di <i>template engine</i> multi-lingua.
40	24/08/2020 - 28/08/2020	Studio della gestione dei comportamenti <i>mobile application</i> in condizioni di mancanza di connettività.
16	31/08/2020 - 01/09/2020	Continuazione ottava settimana.

Tabella 2.1: Tabella riassuntiva delle attività pianificate per il progetto di stage

2.6 Strumenti e tecnologie utilizzate

2.6.1 Tecnologie

HTML

HyperText Markup Language (HTML) è un linguaggio di markup^[g] 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 che porta una sintassi più semplice e un pieno supporto anche a browser più datati.

CSS

Cascading Style Sheets (CSS) è un linguaggio usato per definire la formattazione di documenti HTML, eXtensible Hyper Text Markup Language (XHTML) e eXtensible Markup Language (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 facilitandone la manutenzione. Le specifiche CSS3 sono costituite da sezioni separate dette "moduli" e hanno differenti stati di avanzamento e stabilità.

TypeScript

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

Angular 2+

Angular è un framework^[g] open-source^[g] per la costruzione di applicazioni web con licenza MIT^[g], sviluppato principalmente da Google. Angular è l'evoluzione di AngularJS e 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 vengono eseguite su qualunque piattaforma, inclusi *smartphone* e *tablet*. Inoltre le applicazioni sviluppate in Angular vengono eseguite interamente dal browser web^[g] dopo essere state scaricate dal web server^[g] evitando ulteriori scambi di dati con quest'ultimo ogni volta che c'è una richiesta di azione da parte dell'utente.

Ionic

Ionic è un Software Development Kit (SDK)^[g] open-source^[g] che fornisce strumenti e servizi per lo sviluppo di applicazioni ibride^[g] mobili, desktop e *progressive* basate su moderne tecnologie e pratiche di sviluppo web, che utilizzano tecnologie come CSS3, HTML5 e Syntactically Awesome StyleSheets (Sass). È utilizzabile con qualsiasi framework^[g] per lo sviluppo di applicazioni web. In particolare, le applicazioni ibride^[g] mobili, nonostante siano costruite con tecnologie web, possono essere distribuite tramite *app store* nativi per essere installate sui dispositivi mobili, grazie all'utilizzo di Cordova o Capacitor. Inoltre, Ionic offre componenti grafiche ottimizzate per il *mobile*.

Apache Cordova

Cordova è un framework^[g] open-source^[g] per lo sviluppo di applicazioni *mobile* multi-piattaforma. Esso consente di utilizzare tecnologie standard web come HTML5, CSS3 e JavaScript per lo sviluppo delle applicazioni *mobile*, evitando di utilizzare il linguaggio nativo di ogni piattaforma *mobile*. Inoltre offre Application Program Interface (API)^[g] per accedere ai sensori del dispositivo come ad'esempio la fotocamera. Infatti Cordova

incapsula l'applicazione sviluppata con tecnologie web e la esegue localmente all'interno di un'applicazione nativa che può interagire con le funzionalità del dispositivo.

Protractor

Protractor è un framework^[g] di *test* E2E^[g] per applicazioni Angular e AngularJS. Protractor permette di interagire con l'applicazione in *testing*, come farebbe un utente, attraverso dei metodi messi a disposizione da lui detti "localizzatori" .

Appium

Appium è un framework^[g] open-source^[g] che permette di eseguire in modo automatizzato *script* per testare applicazioni native^[g] o applicazioni web mobile^[g] o applicazioni ibride^[g] su un dispositivo Android^[g] o iOS^[g] utilizzando un API^[g] detta *API WebDriver*.

Cucumber

Cucumber è uno strumento che permette di creare *test* automatizzati con una specifica non ambigua scritta nel linguaggio Gherkin. Gherkin è il linguaggio che Cucumber usa per definire i passi dei *test*. È progettato per essere poco tecnico e leggibile dall'uomo e perciò permette di definire la struttura dei *test* dichiarando i vari passi da eseguire. Proprio per questo Cucumber supporta quindi la behavior-driven development (BDD). Infine Cucumber documenta come si comporta effettivamente il sistema.

Selenium

Selenium è un framework^[g] open-source^[g] che viene utilizzato per automatizzare i *test* effettuati sui browser web^[g] cioè le applicazioni web vengono testate utilizzando un qualsiasi browser web^[g]. Selenium permette di eseguire *test* scritti in vari linguaggi di programmazione, come C# , Groovy , Java , Perl , PHP , Python , Ruby, Scala e JavaScript. Grazie all'utilizzo di un set di API^[g] detto *WebDriver*, i *test* possono essere eseguiti sulla maggior parte dei browser web^[g] moderni.

Npm

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

2.6.2 Strumenti

WebStorm

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

Jira Software

Jira Software è un software proprietario che consente il *bug tracking* e la gestione dei progetti agile sviluppato da Atlassian.

Jira Confluence

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

GitLab è una piattaforma web open-source^[g] che permette la gestione di *repository Git* e di funzioni *trouble ticket*.

2.7 Motivazioni personali

Attraverso la partecipazione all'iniziativa di StageIT, organizzata dall'Università di Padova e da Assindustria venetocentro, ho avuto l'opportunità di 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 che offrisse le seguenti possibilità:

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

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

Questo perché la loro proposta presentava in modo molto esplicito e significativo i tre punti elencati in precedenza. Infatti questa esperienza mi ha permesso di migliorare le mie conoscenze e competenze nell'utilizzo di Angular, imparando a utilizzare i metodi offerti da esso in modo più efficiente. Inoltre, ho avuto la possibilità di sviluppare un'applicazione *mobile* grazie all'utilizzo di Ionic e Cordova.

Un altro aspetto importante che ho apprezzato è che quest'azienda gestisce i propri progetti utilizzando la metodologia agile SCRUM^[g], una tematica che mi interessava approfondire per poter essere una valida alternativa al modello incrementale appreso durante il progetto del corso di Ingegneria del Software.

Infine l'azienda è una realtà giovane nata da meno di cinque anni fatta da persone giovani in cui potevo inserirmi facilmente ed in modo positivo ma anche propositivo.

3 | ARCHITETTURA DEL SISTEMA AWMS

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

3.1 Descrizione

Come scritto precedentemente, per realizzare l'applicazione *mobile* è presente un'architettura^[g] di sistema che permette la comunicazione tra la piattaforma AWMS e l'applicazione *mobile* con Azzurra.

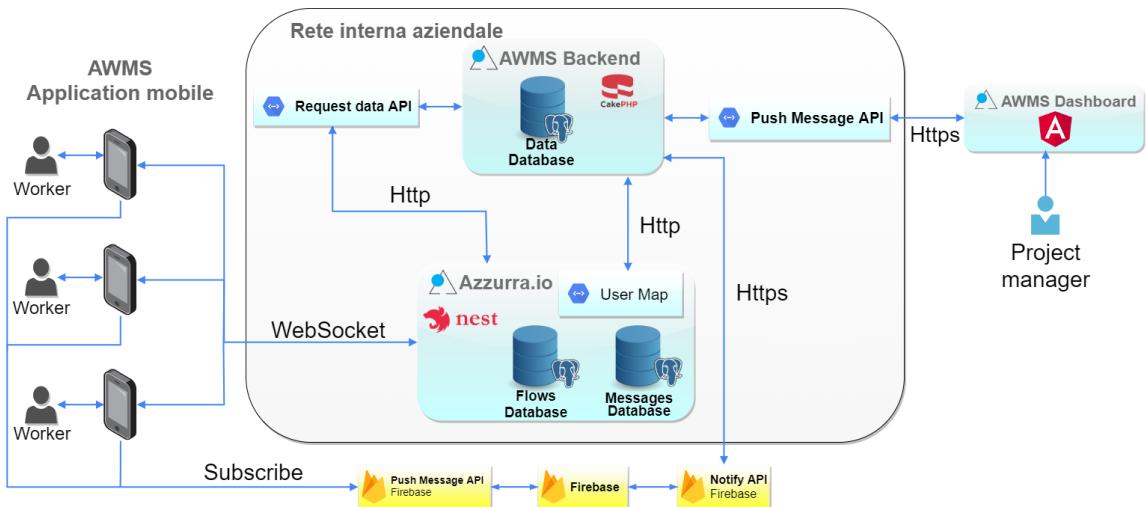


Figura 3.1: Architettura di sistema AWMS

La figura precedente illustra come è composta l'architettura^[g] in cui ogni componente verrà descritta nelle successive sotto sezioni.

3.1.1 AWMS Dashboard

È il pannello di controllo attraverso il quale un plant manager^[g] interagisce con la piattaforma AWMS per 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 attraverso il framework^[g] Angular. La *dashboard* per comunicare con il back-end^[g] utilizza delle API^[g] da esso esposte. Quindi, per ragioni di sicurezza, back-end^[g] e applicazione web, cioè il front-end^[g], comunicano attraverso API^[g] e fanno uso del protocollo di comunicazione Hyper Text Transfer Protocol over Secure

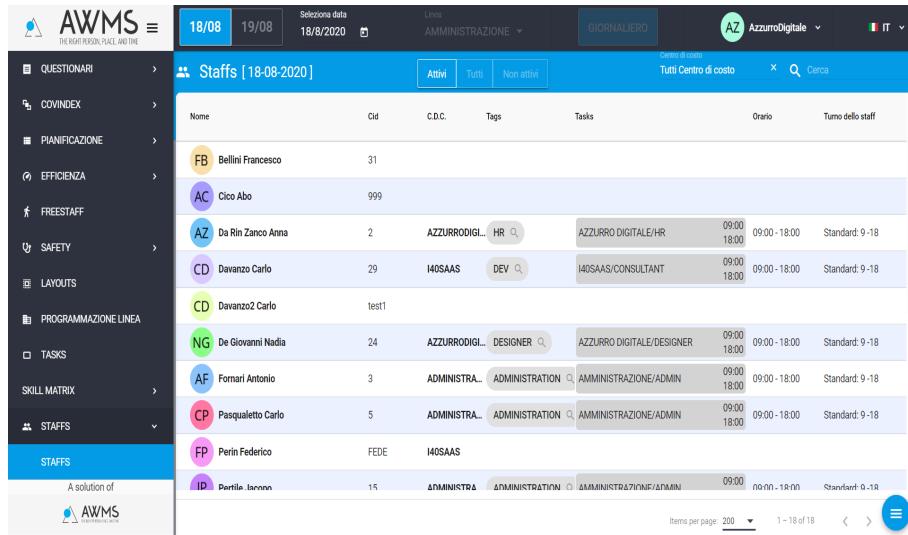


Figura 3.2: Schermata di AWMS Dashboard

Socket Layer (HTTPS)^[g] che rende sicura la comunicazione.

Nella Figura 3.1 viene mostrato il caso in cui il front-end^[g] utilizza API^[g] per l'invio di una notifica push^[g], questo perché è previsto che, quando il plant manager^[g] sceglie il lavoratore più idoneo per un certo lavoro, quest'ultimo deve essere avvisato. Perciò sarà compito del front-end^[g] avvisare il back-end^[g] la presenza di una nuova assegnazione e che questa 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 back-end^[g] del sistema. È stato sviluppato usando la tecnologia CakePHP, un framework^[g] per lo sviluppo di applicazioni web scritto in PHP. Al suo interno risiede il database^[g] che contiene tutte le informazioni sui lavoratori e tra questi i dati da mostrare nei messaggi di Azzurra come ad esempio il piano di lavoro che ha il lavoratore in uno specifico giorno. Il database^[g] utilizza come Database Management System (DBMS)^[g] PostgreSQL.

Come scritto nel punto precedente, per comunicare con AWMS Backend, vengono esposte delle API^[g] per la comunicazione. Esiste un API^[g] per l'invio di una notifica push^[g] e un API^[g] utilizzata da Azzurra.io per la richiesta di informazioni sul lavoratore, necessarie per completare il flusso di conversazione. Quindi questa API^[g] permetterà di richiedere dati al back-end^[g] che li cercherà nel suo database^[g] interno. Se l'interrogazione al database^[g] dà esito positivo ritornerà le informazioni richieste a Azzurra.io. Il back-end^[g] 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 la comunicazione avviene attraverso il protocollo di comunicazione Hyper Text Transfer Protocol (HTTP)^[g]. Il back-end^[g] ha la possibilità di comunicare direttamente con Azzurra.io quando deve inviare una notifica push^[g]. Sarà poi onore di Azzurra.io inviare la notifica push^[g] ai destinatari ma solo a

quelli attivi, cioè quei utenti che hanno una connessione aperta con Azzurra.io. Per i destinatari che non sono attivi, quelli che non hanno una connessione aperta con Azzurra.io, l'invio della notifica verrà fatto utilizzando i servizi offerti da firebase^[g] la cui comunicazione avviene tramite HTTPS^[g].

La gestione dell'invio di una notifica push^[g] verrà comunque trattata 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^[g] Azzurra ed è sviluppata attraverso il framework^[g] NestJS. Al suo interno ha due database^[g] con DBMS^[g] PostgreSQL. Il primo contiene i flussi di conversazione che indicano al bot^[g] la sequenza di passi che deve seguire durante la conversazione con l'utente umano, il secondo permette di memorizzare i messaggi fatti tra il bot^[g] Azzurra e l'utente. La scelta di adottare quest'ultimo database^[g] è dettata dalle seguenti motivazioni:

- * Per mantenere lo stato della conversazione, cioè se l'utente decide di fermare la conversazione e continuarla in un secondo momento, l'utente potrà continuare la conversazione da dove l'aveva lasciata. Il mantenimento dello stato non avviene per un tempo illimitato ma dura per un massimo di un'ora, successivamente verrà cancellato;
- * Per una migliore *user experience* si è scelto di mostrare i messaggi delle conversazioni precedenti nel caso ci siano state delle conversazioni in precedenza. In questo modo se l'utente ha bisogno di un'informazione che ha già chiesto prima, potrà semplicemente controllare 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^[g] che permettono di aprire una connessione tra i due e mantenere sempre aggiornati i dati, come ad esempio la struttura dei flussi di conversazione. Per tenere traccia degli utenti connessi con Azzurra.io tramite l'applicazione *mobile*, viene utilizzata una mappa chiave-valore, interna ad Azzurra.io, denominata *User Map*. Grazie ad essa Azzurra.io potrà sapere quali utenti destinatari di una notifica push^[g] sono attivi e quali no in modo da rispondere alle richieste del back-end^[g] quando viene richiesto l'invio della di una notifica push^[g].

Inoltre Azzurra.io è una componente strategica principalmente per due motivi:

- * Quando si vuole aggiungere un nuovo flusso conversazionale o modificare un flusso già esistente, se non esistesse Azzurra.io, sarebbe salvato nell'applicazione. Ciò comporterebbe la necessità di aggiornare l'applicazione *mobile* e di effettuare una nuova pubblicazione nel Play Store per i dispositivi Android^[g] e nell'Apple Store per i dispositivi iOS^[g] ad ogni aggiunta o modifica dei flussi. Grazie all'esistenza di Azzurra.io ciò viene grazie al database^[g] dedicato alla memorizzazione dei flussi conversazionali che permettono di aggiungere o modificare un nuovo flusso all'interno del database^[g]. Inoltre, grazie alla connessione tramite WebSocket^[g], qualunque modifica o aggiunta viene subito recepita dell'applicazione *mobile*;
- * Per evitare che vengano fatte un numero elevato di richieste al back-end^[g] si è deciso di distribuire le informazioni in diverse componenti della rete. Il bot^[g], infatti, 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 back-end^[g]. Il back-end^[g] però viene contattato quando il bot^[g] Azzurra ha bisogno di dati sul lavoratore da mostrare, ciononostante questa richiesta è fatta inizialmente a Azzurra.io che si prende carico di richiedere le informazioni al back-end^[g] e di ritornarle all'applicazione.

Quindi il back-end^[g] 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 back-end^[g].

3.1.4 Applicazione mobile AWMS

L'ultimo componente dell'architettura^[g] è l'applicazione *mobile* AWMS sviluppata attraverso il framework^[g] Angular2+ e Ionic. In realtà essa è un'applicazione ibrida^[g] perché sviluppata con tecnologie web, nello specifico HTML, CSS e TypeScript, ed eseguita in un dispositivo *mobile* grazie a Ionic e a Cordova. Al suo interno risiede il bot^[g] Azzurra e altre due sezioni: Questionario e Profilo.



Figura 3.3: Sezione Questionario

La Figura 3.3 mostra la sezione Questionario nei suoi tre 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, permettono all'applicazione 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 per la propria salute, l'applicazione mostrerà una faccina rossa, come si può vedere dalla terza immagine della Figura 3.3.

Nel questionario da compilare viene richiesta l'eventuale presenza di sintomi di malattie come mostra la prima immagine della Figura 3.4.

Successivamente viene richiesto se le persone vicino a noi hanno avuto dei sintomi di malattie come mostrato nella seconda immagine della Figura 3.4 ed infine viene

richiesto in quali luoghi si è stati come mostrato nella terza immagine della Figura 3.4. Una volta terminato il questionario, l'applicazione elaborerà le risposte date e mostrerà



Figura 3.4: Schede del questionario sulla salute

l'esito sulla nostra salute. Nel caso in cui l'esito sia positivo verrà mostrata la prima immagine della Figura 3.5, viceversa verrà mostrata la seconda immagine della Figura 3.5.



Figura 3.5: Schede dell'esito del questionario sulla salute

Il risultato viene quindi 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. Al momento non danno nessun particolare beneficio ma in futuro è prevista l'implementazione di una qualche ricompensa. Vengono mostrate le proprie informazioni personali cliccando il tasto "Informazioni personali" come mostrato in Figura 3.6 ed è possibile cambiare la password d'accesso cliccando il

bottone "Gestione password". Cliccando il bottone "Istruzione di utilizzo", visualizzato sempre nella Figura 3.6, è possibile accedere ad una breve guida su come utilizzare l'applicazione. Nel bottone "Normativa privacy" è possibile visionare la normativa sulla tutela della *privacy* General Data Protection Regulation (GDPR)^[g] mentre nel bottone "Titolare trattamento" viene indicato da chi vengono trattati i dati inseriti.

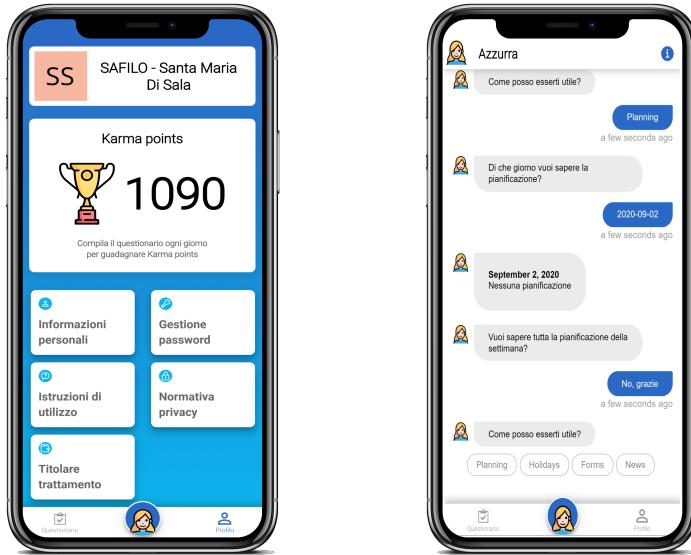


Figura 3.6: Sezione Profilo e Chat con Azzurra

Nella sezione Azzurra, è presente la *chat bot*^[g] 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^[g] Azzurra capisce quali risposte e domande fare all'utente umano. Come detto precedentemente, la comunicazione con Azzurra avviene attraverso WebSocket^[g] che permette di tenere aggiornati i flussi conversazionali ricevuti da Azzurra nel caso in cui subiscano modifiche.

Azzurra è quindi un bot^[g] che aiuta il lavoratore a:

- * Visualizzare il proprio turno di lavoro;
- * Visualizzare i propri permessi lavorativi o richiederne di nuovi;
- * Visualizzare avvisi da parte dell'azienda;
- * Sapere qual'è il menu del giorno della mensa aziendale;
- * Effettuare prenotazioni di un posto a sedere in una sala riunioni, visualizzare le proprie prenotazioni e utilizzare un scannerizzatore QR code^[g] per riscattare il posto prenotato.

Gestione modalità offline

Nell'ambito dello sviluppo di un'applicazione *mobile* un aspetto importante è la gestione della modalità *offline*. Per gestione della modalità *offline* si intende permettere agli utenti di continuare ad utilizzare l'applicazione in assenza di connettività e quindi

l'impossibilità di comunicare con il back-end^[g]. Quali e quante funzionalità e con quali e quante limitazioni alle stesse si verificano quando si è *offline*, è una scelta di progettazione e comporta uno sforzo più o meno significativo, visto che esistono diverse modalità per ottenere quanto richiesto. Nel caso dell'applicazione *mobile* AWMS in assenza di connettività sono state scelte le seguenti limitazioni e funzionalità:

- * Quando c'è assenza di connettività viene avvisato l'utente di tale evento attraverso un messaggio che comparirà sopra l'*header* dell'applicazione;
- * Vengono garantite un insieme minimo di funzionalità anche in caso di assenza di connettività. Nello specifico viene garantita all'utente la possibilità di spostarsi tra le sezioni dell'applicazione accedendo solo ad alcune pagine informative quali la "Informazioni personali", la "Privacy policy", le "Istruzioni di utilizzo" dell'applicazione.

La gestione della modalità *offline* verterà principalmente su due aspetti:

- * Stato della connessione;
- * Persistenza locale dei dati.

Stato della connessione

Poter sapere lo stato della connessione per il funzionamento dell'applicazione *mobile* è fondamentale considerando anche che la connettività cambia nel tempo. Infatti può accadere che l'utente sia connesso all'inizio ma ad un certo punto possa perdere la connessione. Vi è quindi la necessità di sapere non solo lo stato della connessione all'inizio ma anche i cambiamenti nello stato della connettività. Solitamente per determinare lo stato della connessione si accede direttamente allo strato nativo del *device*. Per fare ciò è stato scelto di utilizzare il *plugin* di Cordova, Network Information, che, oltre a determinare lo stato della connessione, è in grado di rilevare cambiamenti dello stato in maniera agevole.

Persistenza locale dei dati

La possibilità di salvare i dati in locale, permette di garantire le esecuzioni di alcune funzioni anche in assenza di connettività. Esistono diverse possibilità per la memorizzazione locale dei dati, più o meno sicure o efficienti.

Per l'applicazione *mobile* AWMS è stato scelto di utilizzare SQLite, un DBMS^[g] file-based memorizzato sul *device* sul quale viene installata l'applicazione. La capacità del database^[g] (dimensione massima) dipende dalla memoria a disposizione nel *device* offrendo quindi un buon supporto per la memorizzazione e l'esecuzione delle comuni operazioni sui dati.

Per l'applicazione *mobile*, essendo un'applicazione ibrida^[g] costruita con Ionic, è stato scelto di utilizzare Ionic Storage composto a sua volta di un *plugin* di Cordova, per la gestione dei database^[g] SQLite.

Utilizzando Ionic Storage è possibile quindi memorizzare delle coppie chiave-valore dove il valore può essere di qualsiasi tipo, dalle semplici stringhe o interi fino a degli oggetti JavaScript Object Notation (JSON)^[g]. I dati memorizzati localmente rimarranno disponibili sino alla disinstallazione dell'applicazione o alla formattazione del *device*.

3.2 Operazioni

Nella seguente sezione verranno descritte le principali operazioni tra le varie componenti dell'architettura^[g].

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^[g]. Grazie a ciò si ha un canale di comunicazione a due vie in cui sia l'applicazione *mobile* e sia Azzurra.io possono inviare dati o richieste. Inoltre 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^[g], risulta essere più efficiente e performante rispetto al pooling^[g] in quanto il server^[g] non viene continuamente contatto da inutili richieste.

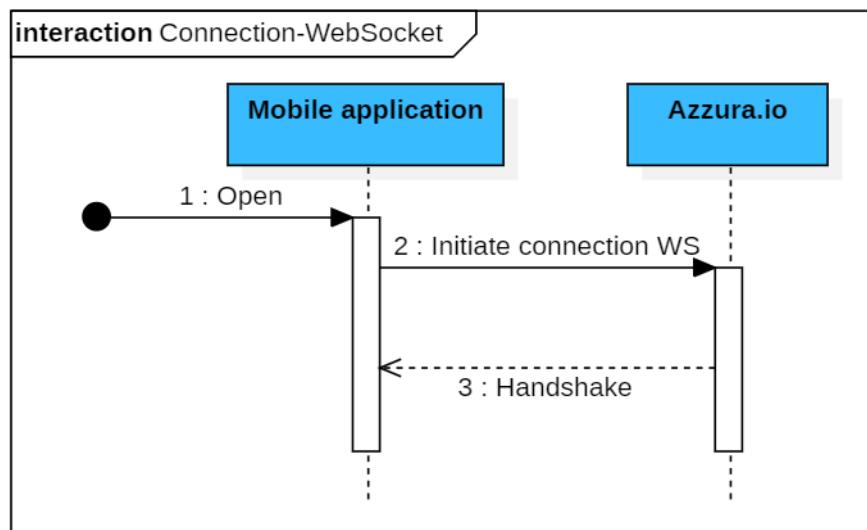


Figura 3.7: Sequence diagram per la creazione di una connessione attraverso WebSocket

Nella Figura 3.7 viene mostrato come avviene la creazione di una connessione tra l'applicazione *mobile* e Azzurra.io. I passi perciò sono:

1. L'applicazione *mobile* viene aperta dall'utente e cercherà da subito di mettersi in contatto con Azzurra.io creando una connessione;
2. Per avviare una connessione WebSocket^[g], viene inviata una richiesta HTTP^[g] a Azzurra.io(server^[g]). Negli *headers* dell'intestazione di tale richiesta viene specificata un'operazione di tipo *Upgrade* ad indicare che l'applicazione *mobile* (client^[g]) vuole aggiornare la connessione ad un protocollo diverso, in questo caso a WebSocket^[g]. 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 soprastante rappresenta un esempio di *WebSocket handshake request* inviata dal client^[g] al server^[g]. Il client^[g] inserisce una stringa casuale codifica in base64^[g] nel campo *Sec-WebSocket-Key* alla quale viene poi aggiunta una stringa fissa.

3. Se il server^[g], in questo caso Azzurra.io, supporta la connessione tramite WebSocket^[g] allora risponde 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 ed infine codificando tutto in base64^[g]. Una volta arrivata la risposta al client^[g], esso controllerà se contiene la stringa corretta. Tutte queste operazioni hanno lo scopo di evitare l'apertura di più connessioni multiple senza dare alcuna garanzia di autenticazione. Nel seguente codice viene riportato un esempio di risposta da parte del server^[g].

```

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 indesiderati viene usata una variante del WebSocket^[g] detta *Secure WebSocket (WSS)* che utilizza il protocollo HTTPS^[g] al posto del protocollo HTTP^[g]

3.2.2 Recupero di un flusso conversazionale

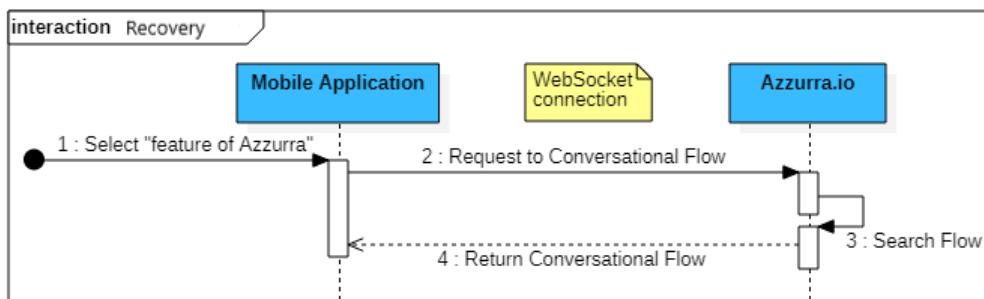


Figura 3.8: Sequence diagram per il recupero di un flusso conversazionale

Il bot^[g] 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^[g] dedicato di Azzurra.io. Quando viene richiesto di fare un'operazione o si inizia per la prima volta un'interazione con Azzurra, essa ha bisogno di uno specifico flusso di conversazione e, 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 l'utente richiede una specifica funzionalità come la visualizzazione del piano di lavoro, Azzurra dovrà richiedere a Azzurra.io il *flow* dedicato alla specifica funzionalità. Come mostrato nella Figura 3.8 si hanno i seguenti passi:

1. L'utente interagisce con il bot^[g] Azzurra e richiede una funzionalità oppure l'utente interagisce con Azzurra per la prima volta;
2. Il bot^[g] 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 e Azzurra sa sempre qual'è quello del flusso di cui ha bisogno;
3. Azzurra.io cerca nel proprio database^[g] se contiene il *flow* richiesto;
4. Se la ricerca da esito positivo Azzurra.io ritornerà il flusso che verrà eseguito dal Flow Engine di Azzurra proseguendo 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 che però non sono salvate né nell'applicazione *mobile* né in Azzurra.io ma nel back-end^[g]. La Figura 3.9 mostra

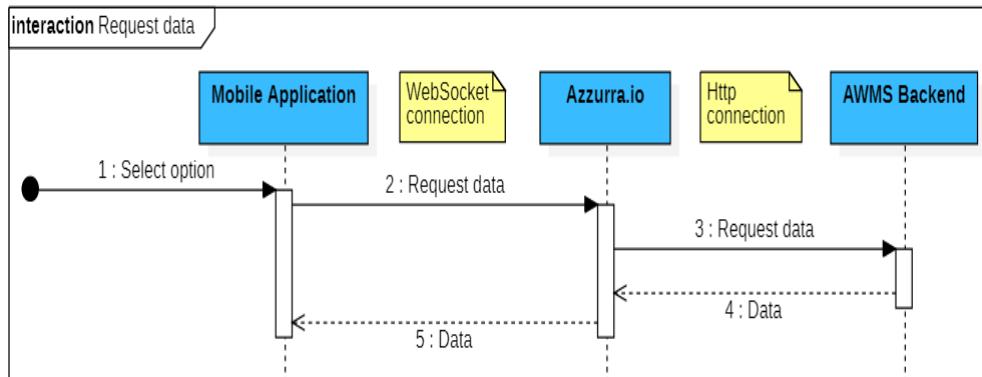


Figura 3.9: Sequence diagram per il recupero dei dati sul lavoratore

come Azzurra ottiene i dati:

1. L'utente chiede una qualche operazione ad Azzurra che necessita contattare il back-end^[g] 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 il back-end^[g] attraverso una richiesta HTTP^[g], per richiedere i dati richiesti da Azzurra;
4. Se la richiesta va a buon fine il back-end^[g] ritornerà i dati ad Azzurra.io che li ritornerà a sua volta ad Azzurra.

Per l'invio dei dati che devono essere salvati sul database^[g] del back-end^[g] come l'inserimento di una nuova assenza, il procedimento sarà analogo alla richiesta dati e 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 plant manager^[g] abbia bisogno di mandare una comunicazione a uno o più lavoratori è stata prevista la possibilità di inviare delle notifiche contenuti la comunicazione.

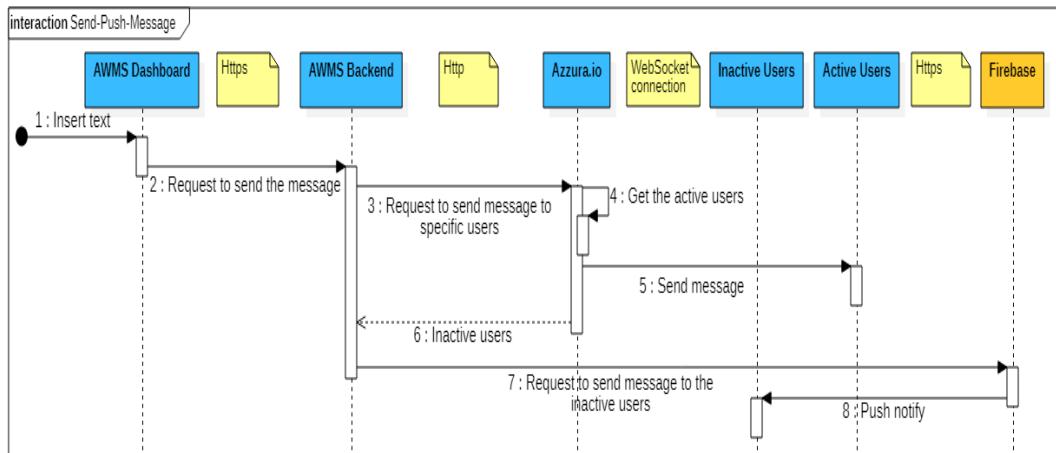


Figura 3.10: Sequence diagram per l'invio di notifiche push

Nella Figura 3.10 viene mostrato come avviene l'invio delle notifiche, di seguito ne vengono descritti i passi:

1. Il plant manager^[g] crea il messaggio da inviare attraverso l'interazione con la *dashboard* di AWMS;
2. La *dashboard* invia al back-end^[g] il messaggio da inviare e a quali utenti registrati nel sistema mandarlo e i codici identificativi dei destinatari;
3. Il back-end^[g] si connette ad Azzurra.io e successivamente invia il messaggio e la lista degli utenti a cui Azzurra.io deve mandare la notifica. Quindi l'invio di quest'ultima viene delegato a Azzurra.io;
4. Azzurra.io controlla sulla sua tabella *User Map* quali tra gli utenti che devono ricevere il messaggio sono attualmente connessi al *socket*. Si specifica inoltre che vengono considerati utenti connessi o attivi tutti quelli che sono connessi al

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

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

4 | AZZURRA FLOW ENGINE

Nel seguente capitolo verrà illustrata la struttura di un flusso conversazionale ed il funzionamento del Azzurra Flow Engine con la conseguente generazione dei messaggi da parte del bot^[g] Azzurra e dell'utente umano. Infine verrà trattata la gestione dell'internazionalizzazione

4.1 Cos'è

Un elemento cardine dell'architettura^[g] del sistema AWMS è Azzurra Flow Engine. Esso è un motore conversazionale in grado di ricevere in input flussi conversazionali, implementati attraverso configurazioni JSON^[g], che risiedono nel database^[g] di Azzurra.io. Essi vengono mandati in input ad Azzurra Flow Engine quando il bot^[g] Azzurra ne fa richiesta. Questi file sono codificati secondo una certa struttura fatta dai cosiddetti blocchi conversazionali e da altri campi illustrati in seguito. Tornando su Azzurra Flow Engine, esso riceve in input la configurazione JSON^[g] e grazie ai metodi che ha disposizione è in grado di interpretare le configurazioni JSON^[g] ricevute e di generare i messaggi che il bot^[g] Azzurra deve far visualizzare all'utente nella *chat*.

4.2 Flussi conversazionali

I flussi di conversazione o conversazionali sono degli elementi fondamentali per la conversazione tra il bot^[g] e l'utente umano. Essi sono delle configurazioni in JSON^[g] dove ogni configurazione contiene un flusso di conversazione e ogni flusso è un possibile ramo di conversazione che può essere fatto tra il bot^[g] Azzurra e l'utente umano. Ogni configurazione contiene perciò dei particolari comandi che permettono al bot^[g] 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 ai programmi *software*, cioè una funzione detta *main* 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 ci sono:

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

4.2.1 Shortcuts “shortcuts”

Il campo shortcuts è dedicato alle cosiddette "scorciatoie" ovvero la possibilità di visualizzare un menu dove vengono visualizzate tutte le funzionalità offerte dal bot^[g] 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: Campo che può essere di tipo *string* e contenere 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 rappresentato. 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 *User Interface* 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 per la conversazione 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 caratteristiche uniche ma anche comuni in quanto tutti i tipi 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

BLOCK è il blocco di conversazione attraverso il quale tutti blocchi ereditano delle caratteristiche comuni della sua struttura. Di fatto BLOCK non può essere utilizzato e, quindi, può essere paragonato a una classe astratta nell'ambito della programmazione ad oggetti in cui 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* che 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 in modo random che 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 “conversazionale” su cui salvare eventuali valori di input inseriti dall'utente: il Flow Engine quindi, attraverso dei metodi specifici, ha la capacità di salvare tutte le scelte fatte dall'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 che verranno descritti successivamente:

- **BUTTONS;**
- **ITEMS;**
- **PICKER;**
- **TIMEPICKER;**
- **DATEPICKER;**
- **CALENDAR;**
- **QRSCANNER.**

* **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 rimanendo in attesa di una risposta dell'utente. Infine esegue il comando collegato alla scelta effettuata dall'utente dirottando la conversazione al blocco successivo.

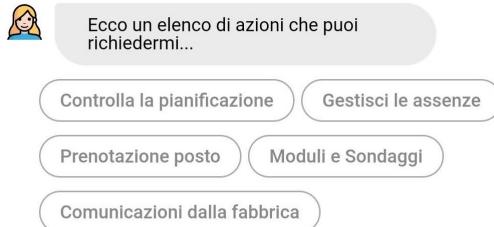


Figura 4.2: Esempio di messaggio prodotto da un blocco di tipo ASK

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;
- * **sourceType:** Parametro che indica la modalità di utilizzo della variabile contenuta nel campo source. Sono previste due modalità:
 - **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 buttoni da far visualizzare a video. Il tipo LIST viene usato quando il campo category ha il valore ASK in modo da poter creare per ogni scelta il giusto campo target attraverso l'oggetto di tipo BlockItem;

- **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 i dati contenuti nella variabile devono essere formattati in una struttura del tipo chiave valore. Il tipo VARIABLE viene usato quando il campo category ha il valore MENU e quando i dati da utilizzare per creare i bottoni sono stati ricevuti come risposta a una richiesta al back-end^[g] quindi, sono memorizzati in una variabile, la stessa indicata nel campo source. Un possibile esempio di utilizzo è il seguente; viene richiesto quali posti a sedere sono liberi per una prenotazione, attraverso una richiesta al back-end^[g] viene ricavato tale dato e salvato in una variabile indicata in source, viene poi per ogni posto libero creato un bottone che l'utente può cliccare per indicare la sua scelta. Tutte le scelte possibili porteranno al blocco che si occuperà di comunicare la scelta al back-end^[g].
- * **source:** Parametro che contiene il nome di una variabile a cui fare riferimento per prendere i dati da utilizzare per costruire le possibili scelte che può fare l'utente; essi graficamente vengono rappresentati come dei pulsanti rettangolari. Se non si fa riferimento a nessuna variabile o sourceType ha valore uguale a LIST, 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 arrotondati.

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, essa viene eseguita e una volta conclusa il bot^[g] risponderà all'utente con il risultato ricavato precedentemente.



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

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

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

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. Viene perciò costruito un *template* del testo del prossimo messaggio da mostrare.

JUMP

Il blocco di conversazione JUMP permette di cambiare il flusso conversazionale ed eseguirne uno altro. In termini tecnici si passa da un JSON^[g] di configurazione ad un altro dove ogni configurazione in JSON^[g] contiene uno specifico flusso di conversazione. Perciò, grazie a JUMP, si può "saltare" da un flusso di conversazione a un altro.

Nel campo target in questo caso 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^[g] (l'applicazione *mobile*) può richiedere l'esecuzione di chiamate ad API^[g] (interne o esterne). Attraverso un WebSocket^[g], che mantiene una connessione tra il bot^[g] e Azzurra.io, quest'ultima richiama a sua volta delle API^[g] 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^[g] 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^[g].
Ha i seguenti campi:
 - * **url**: Contiene l'indirizzo Uniform Resource Locator (URL)^[g] 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^[g];
 - * **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 essere utilizzati all'interno dei blocchi per svolgere un'azione di supporto affinché si possa raggiungere ciò per cui sono stati realizzati i blocchi di conversazione stessi.

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

Widget

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

- * **BUTTONS:** Genera dei bottoni arrotondati;

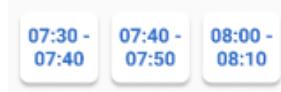


Figura 4.4: Rappresentazione grafica dei buttons

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

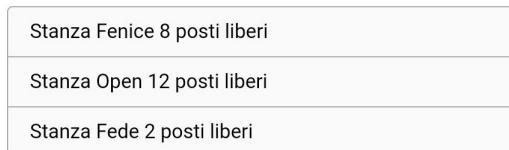


Figura 4.5: Rappresentazione grafica degli items

- * **PICKER:** Genera, attraverso il componente “ion-picker” di Ionic una finestra di dialogo dove si può selezionare una opzione tra quelle proposte;

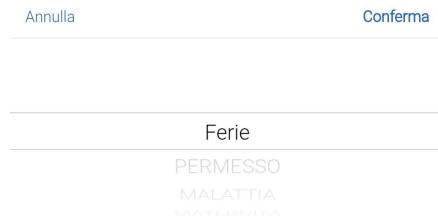


Figura 4.6: Rappresentazione grafica del picker

- * **TIMEPICKER:** Analogico al PICKER solo che l'opzione da scegliere è l'orario che si vuole selezionare;

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



Figura 4.7: 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 di decodificare i codici QR code^[g] tutto ciò grazie al *plugin* QR Scanner di Cordova.

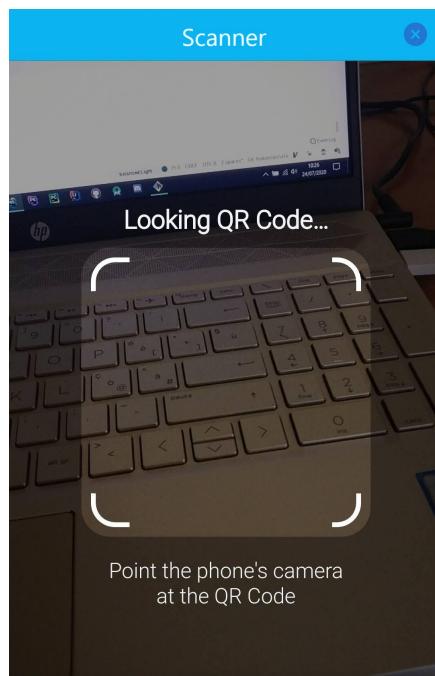


Figura 4.8: Rappresentazione grafica del QR scanner

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 4.9: 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 un'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

L'Azzurra Flow Engine è quell'elemento in grado di interpretare i dati contenuti nelle configurazioni JSON^[g]. Grazie a esso 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^[g] 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 anche 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 far visualizzare e di ricavare da essi le risposte dell'utente umano.

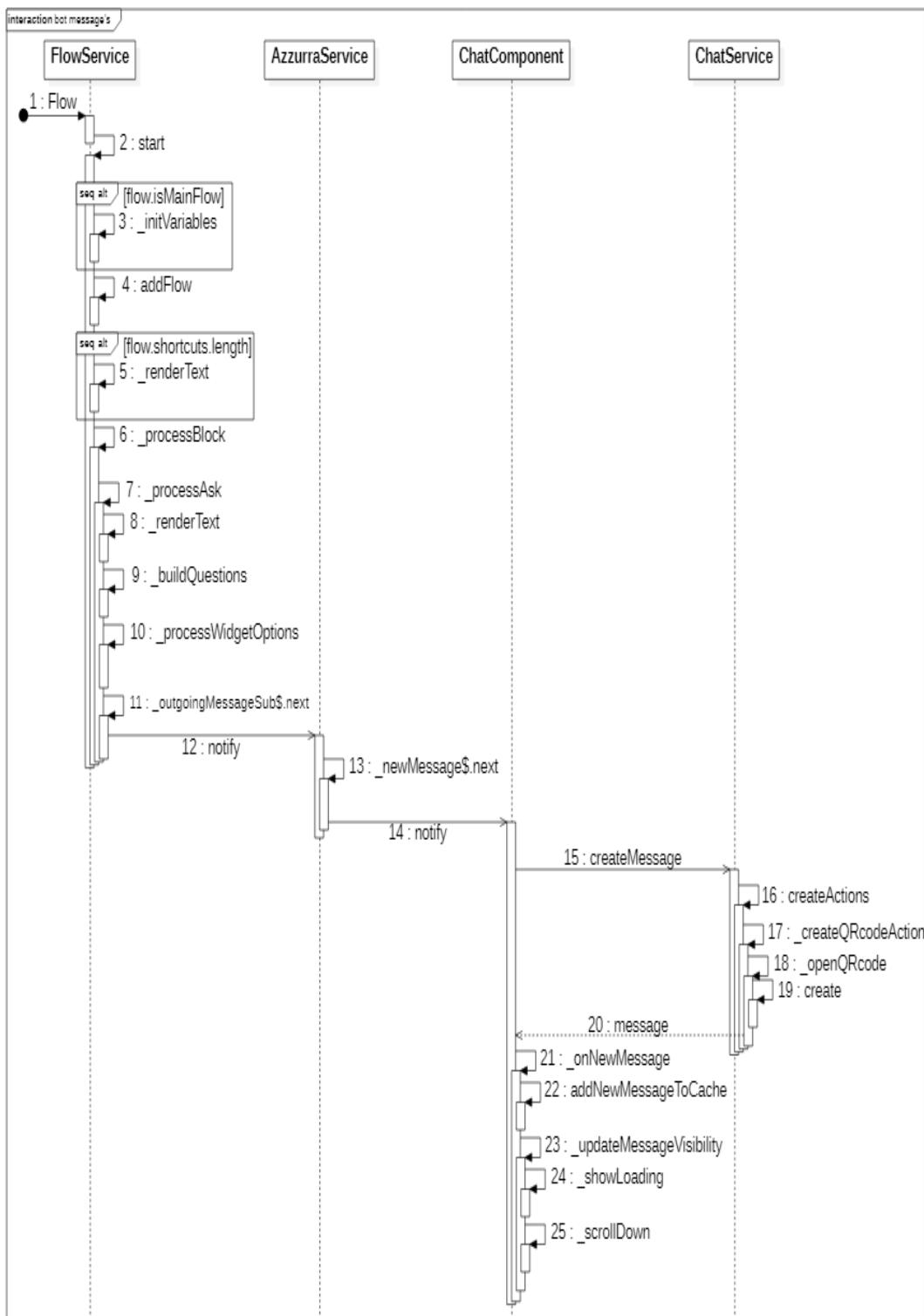


Figura 4.10: Diagramma di sequenza per la generazione di un messaggio di Azzurra

Una volta stabilita la connessione con Azzurra.io, attraverso un WebSocket^[g], inizializzato il Flow Engine e ricevuto la configurazione JSON^[g] contenente il flusso conversazionale, la generazione di un messaggio da parte del bot^[g] 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 variabili conversazionali d'ambiente e uscite dal blocco condizionale viene salvato il *flow* da eseguire attraverso il metodo `_addFlow()` che imposta come flusso corrente da eseguire il flusso ricevuto in input. Viene infine 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 al 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()`. Se presenti vengono creati gli eventuali BlockAttachments. Infine viene emesso il nuovo messaggio creato per il bot^[g] 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 conterà la risposta dell'utente. Analogamente al caso SAY viene generato il testo della domanda. Vengono inoltre generate le possibili scelte attraverso il metodo `_buildQuestions()` il quale controlla il `sourceType` e in base al valore che ha, genera le scelte. Inoltre sono processati anche gli eventuali `widgetOptions`. Infine viene emesso il nuovo messaggio creato per il bot^[g], che rimane in attesa della risposta dell'utente umano quando verrà visualizzato il messaggio;

- * **Caso JUMP:** Viene eseguito il metodo `_processJump()` che riceve in input il blocco corrente e richiama il metodo `_getFlow()` passandogli 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 quale 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 all'esito della risposta si eseguirà il corrispondente blocco successivo.
7. Nella Figura 4.10 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 create le 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 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` emette 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. Nel caso in cui sia del bot^[g] 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^[g];

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 4.10, 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 ed aprire il Widget QRCode;
19. In `_openQRcode()` viene chiamato il metodo `create()` che permette di utilizzare il ModalController di Ionic il quale crea una nuova finestra grafica con dentro la classe che gestisce il lettore di QR code^[g]. Si crea quindi graficamente il Widget. I corrispondenti metodi *open* per DATEPICKER e TIMEPICKER non utilizzano il ModalController ma utilizzano una componente grafica messa a disposizione da Ionic, perciò in questi metodi viene configurato il componente grafico senza richiamare nessuna classe;
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 tramite 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:

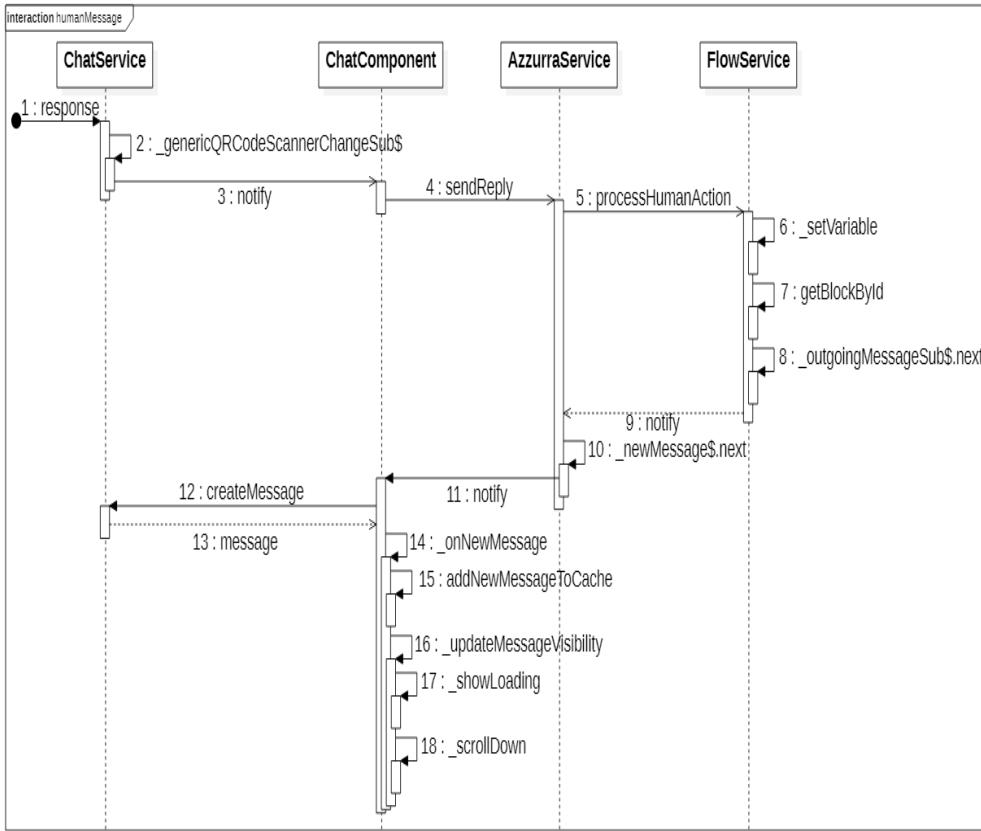


Figura 4.11: Diagramma di sequenza per la generazione di un messaggio dell'utente

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 4.11 viene rappresentato il caso in cui il Widget sia di tipo QR CODE; pertanto, attraverso il metodo next() di Angular, viene emesso un segnale che avvisa a chi e in ascolto su _genericQRCodeScannerChangeSub\$ che è stato emesso un evento da QR CODE 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. In caso affermativo viene impostato come prossimo blocco di conversazione il blocco che ha il codice identificativo uguale a quello contenuto nel target, viceversa 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 è in ascolto su `_outgoingMessageSub$` l'emissione di un nuovo messaggio;
9. Viene notificato agli ascoltatori l'evento descritto al punto precedente;
10. AzzurraService riceve la notifica e tramite il metodo `next()` avvisa e invia la struttura del nuovo messaggio a tutti coloro che sono in ascolto su `_newMessage$`;
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 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^[g] Azzurra.

4.4 Gestione dell'internazionalizzazione di AWMS

Per il sistema AWMS è stata implementata la gestione dell'internazionalizzazione (i18n). Infatti per rendere tutto il sistema, sia lato back-end^[g] sia lato Azzurra, adattabile a più lingue, sono stati definiti un insieme di testi multi-lingua. Per gestirli è stato progettato di utilizzare un *template engine* al fine di ottenere la parametrizzazione dei testi.

Un *template engine* è un programma che prevede la definizione di blocchi detti *template*, spesso scritti in HTML, che hanno la caratteristica di avere dei segnaposto detti *placeholders* sostituiti dai dati da visualizzare. In questi blocchi perciò viene definito il testo scritto in una specifica lingua e in quali punti del testo devono essere inseriti i dati per la visualizzazione. Per la sostituzione dei *placeholders* solitamente il *template engine* utilizza file JSON^[g], in cui i segnaposti sono solitamente rappresentati dai campi che compongono l'oggetto salvato nel file JSON^[g], i quali andranno a sostituire i segnaposti con i loro valori che rappresentano i dati da visualizzare. Chiaramente i *template* vengono riutilizzanti anche se i dati visualizzati variano. Vengono perciò definiti *template* di testi "localizzati" cioè scritti in un delle lingue supportate dal sistema AWMS.

Un *template engine* può essere facilmente implementato in JavaScript ma come scelta progettuale è stato deciso di utilizzare *Handlebars*. *Handlebars* è un'estensione del *template engine mustache* perché utilizza le doppie parentesi graffe le quali racchiudono la parola da sostituire per indicare i *placeholders* all'interno dei *template*. È stato scelto di utilizzare *Handlebars* perché garantisce delle performance migliori: permette di compilare i *template* riducendo così il tempo di *rendering* quando un blocco di codice viene creato più di una volta. Inoltre *Handlebars* permette di utilizzare i cosiddetti "Helpers" che eseguono funzionalità che vanno a manipolare la formattazione del testo. Infine una caratteristica avanzata di *Handlebars* è il supporto ai "Partials". Con tale termine si indica la possibilità di innestare *template* l'uno dentro l'altro, permettendone la riusabilità e la realizzazione di codice più ordinato.

Nel back-end^[g] ciascun testo "localizzato" è persistito in apposite tabelle di un *database*.

Nel lato bot^[g] Azzurra, nel campo *text* dei blocchi conversazionali, c'è 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*. Quindi Il Flow Engine, interpretando ciascun blocco conversazionale, effettua un processamento dei testi scegliendo quelli relativi alla lingua dell'utente se presenti.

Sarà poi compito di *Handlebars* "idratare" i *template* sostituendo i *placeholders* in sintassi *mustache* con dati da visualizzare, sempre se presenti.

Durante lo stage un parte del tempo è stata dedicata allo studio di *Handlebars* e alla scrittura dei *template* che in molti casi hanno richiesto l'inserimento dei *placeholders* con la sintassi *mustache*.

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, in comune accordo con il tutor aziendale, di costruire due flussi conversazionali per il bot^[g] 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 ed infine, scannerrizzare un QR code^[g] messo nel posto a sedere, per controllare se il lavoratore può usufruire e riscattare tale posto. C'è quindi bisogno di integrare un lettore di QR code^[g] in Azzurra per poter fare il controllo che consiste nell'aprire la fotocamera e scannerizzare il QR code^[g] che verrà usato per il controllo ed infine 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, vincolante in quanto primario e fondamentale;
 - * **2:** Requisito desiderabile, non strettamente necessario ma che porta valore aggiunto riconoscibile;
 - * **3:** Requisito opzionale, relativamente utile.
 - **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 prodotto;

- * **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 ^[g] 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 desiderata.
R1F7	Il lavoratore deve poter inserire l'ora in cui finisce la prenotazione desiderata.
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.

Tabella 5.1: Tabella del tracciamento dei requisiti

Codice	Descrizione
R1F16	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 in cui finisce la prenotazione.
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 ^[g] del posto a sedere, deve ricevere un messaggio di conferma che lo informa che può usufruire del posto a sedere.
R1F21	Il lavoratore, dopo avere scannerizzato il QR code ^[g] del posto a sedere, deve ricevere un messaggio d'errore che lo informa che 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 in cui termina il 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 ^[g] deve essere usato il <i>plugin</i> 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, sono passato alla progettazione dei flussi. Come spiegato nel capitolo precedente i flussi conversazionali sono un insieme di blocchi che svolgono determinate funzioni a seconda del tipo di appartenenza. 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 vengono visualizzati quali passi deve fare il bot^[8] Azzurra durante la conversazione con l'utente umano. Successivamente ho progettato 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 visualizzato l'insieme dei blocchi che fanno parte del flusso conversazionale DeskBooking. Per comodità si è deciso di rappresentare il flusso attraverso tre diagrammi più piccoli.

La Figura 5.1 rappresenta il ramo del flusso Deskbooking 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^[8];
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 viene chiesta l'ora di inizio per la prenotazione tramite un blocco ASK. L'inserimento dell'ora avviene tramite il TIMEPICKER;
4. Viene rifatta la stessa operazione del punto precedente ma chiedendo la data in cui finisce la 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 ad 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 e 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;

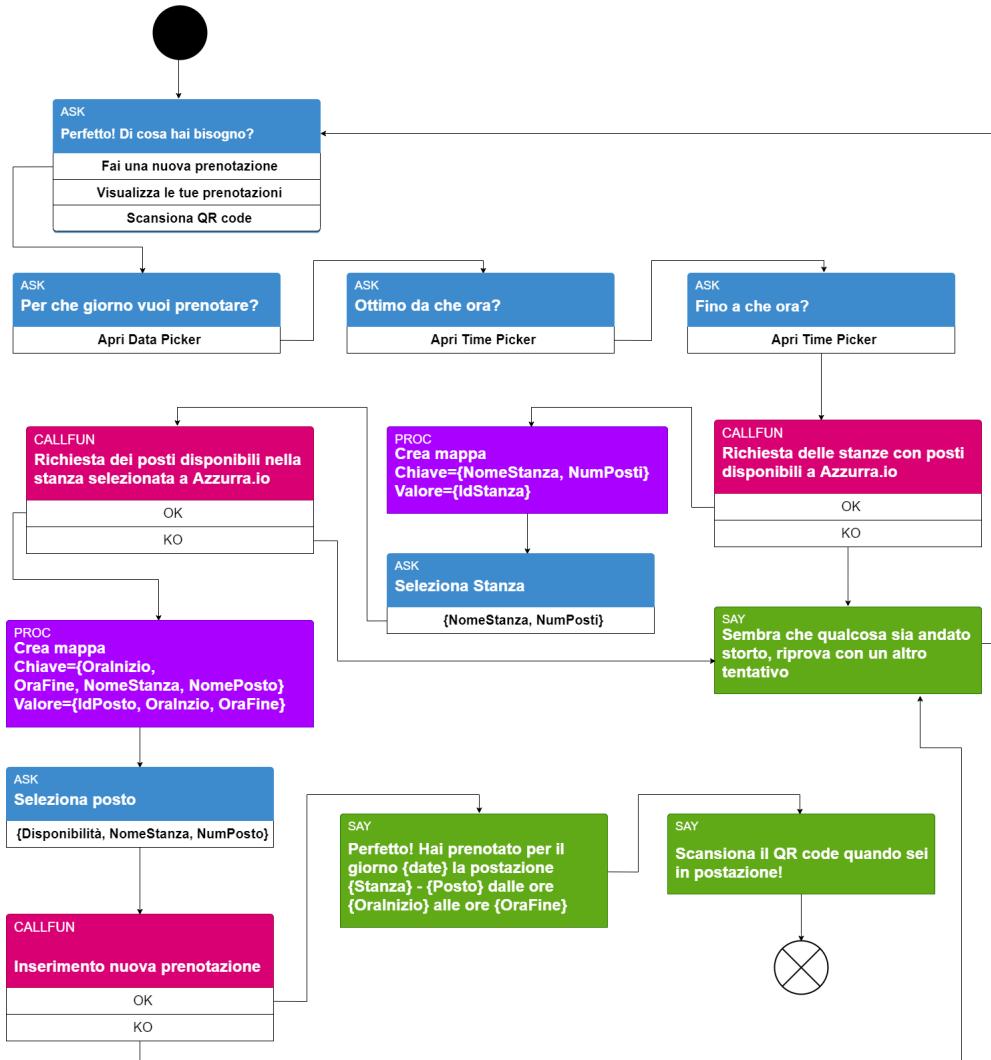


Figura 5.1: Diagramma per l'inserimento di una nuova prenotazione del flusso DeskBooking

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^[g] per riscattare il posto a sedere. Il flusso poi termina.

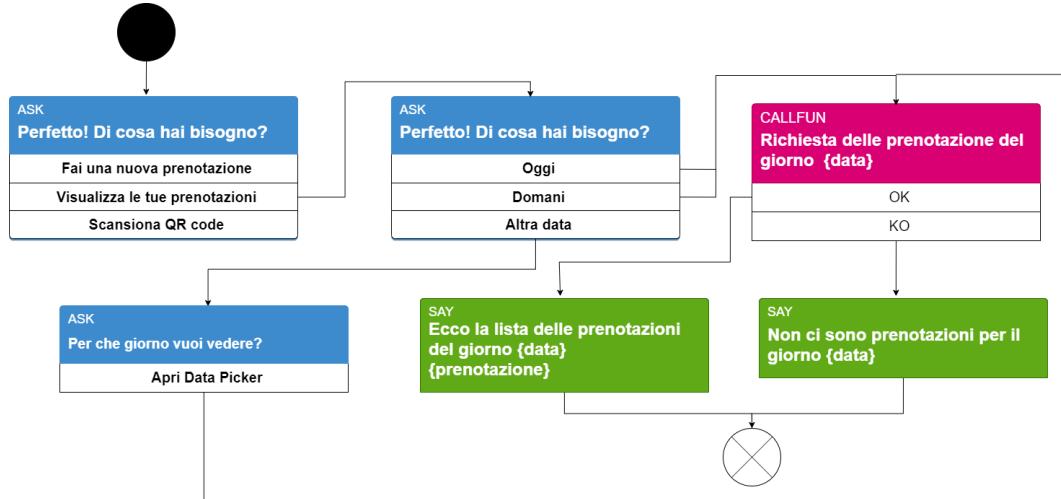


Figura 5.2: Diagramma per la visualizzazione delle prenotazioni del flusso DeskBooking

La Figura 5.2 rappresenta il ramo del flusso Deskbooking 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^[g];
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 ad Azzurra.io per ottenere le prenotazioni della data inserita dall'utente. La richiesta viene fatta attraverso il blocco CALLFUN;
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.

La Figura 5.3 rappresenta il ramo del flusso Deskbooking dedicato allo scansionamento del QR code^[g]. 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^[g];

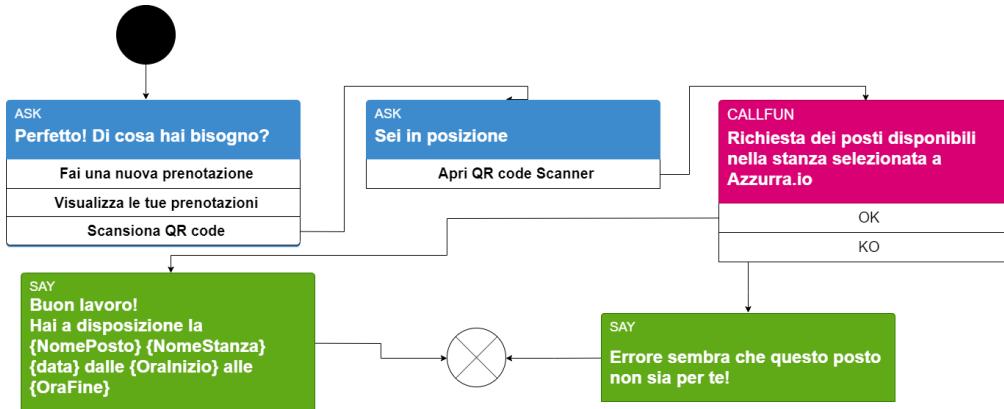


Figura 5.3: Diagramma per lo scansionamento del QR code^[g] del flusso DeskBooking

2. Nel caso in cui l'utente voglia scansionare un QR code^[g] per riscattare il suo posto a sedere prenotato, viene chiesto attraverso un blocco ASK di aprire lo scannerizzatore di QR code^[g]. Viene usato QRSCANNER per leggere il QR code^[g];
3. Viene chiesto ad 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.

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;

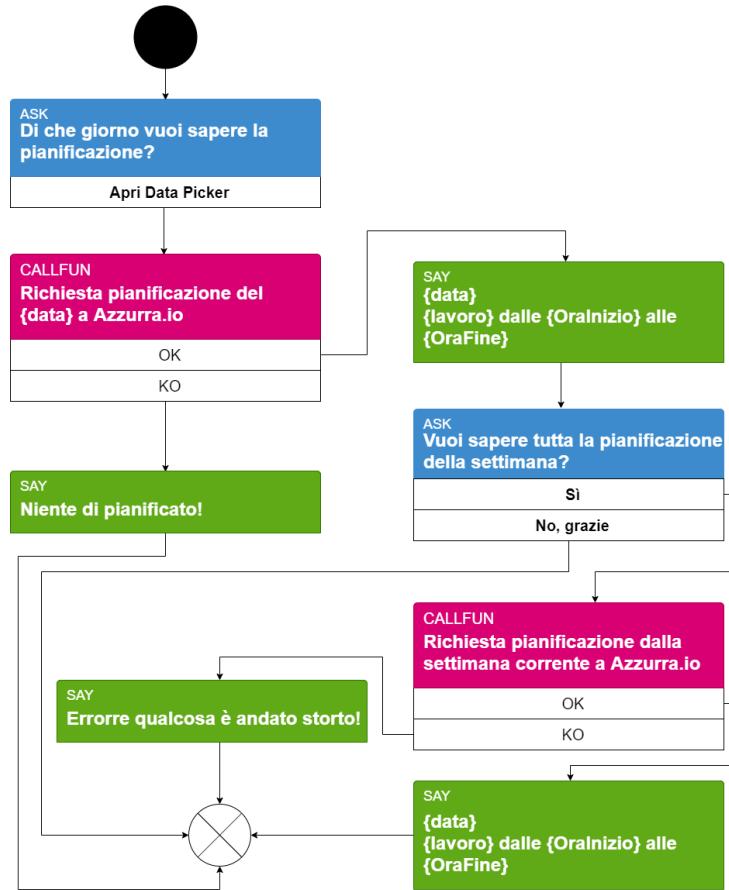


Figura 5.4: Diagramma per la visualizzazione della pianificazione del flusso Planning

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^[g] 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* infatti solo utilizzando Angular si può sviluppare una applicazione web. Si è quindi usato Cordova, un framework^[g] che permette di sviluppare un'applicazione *mobile* multi-piattaforma e offre API^[g] per accedere alle funzionalità native del dispositivo, ad esempio la fotocamera. Cordova infatti incapsula l'applicazione web e la esegue localmente all'interno di un'applicazione nativa^[g] che può interagire con le funzionalità del dispositivo. Per sfruttare le funzionalità di Angular e di Cordova assieme, è stato usato il framework^[g] Ionic che permette di creare un ambiente integrato che semplifica lo sviluppo di applicazioni offrendo anche componenti grafiche ottimizzate per i dispositivi *mobile*.

Per quanto riguarda la codifica, per prima cosa si è implementato una configurazione JSON^[g] 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.

Per poter creare i tre Widget, DATEPICKER, TIMEPICKER e QRSCANNER, nel createActions() ho aggiunto tre metodi per ognuno dei tre Widget, che vengono chiamati da createActions() in base al tipo di Widget da creare. In questi tre nuovi metodi viene impostato il testo che devono mostrare e nel caso dei DATEPICKER e TIMEPICKER viene impostato anche il formato del giorno e dell'ora. Per ognuno di questi metodi ho creato un metodo specifico per ogni Widget che si occupa della creazione e dell'apertura, 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^[g]. Una volta finito di usare la nuova finestra, essa viene chiusa e si ritorna alla finestra precedente che sarà nello stato precedente all'apertura della nuova finestra. Ho dovuto perciò implementare la classe che gestisce il lettore QR code^[g], denominata CameraComponent, dove al suo interno richiama il *plugin* di Cordova, QR Scanner. QR Scanner è un API^[g] che permette di accedere alla fotocamera del dispositivo e di scansionare i QR code^[g]. Vengono quindi definiti due metodi in CameraComponent, un metodo per l'apertura della fotocamera, la lettura del QR code^[g] e la chiusura della fotocamera che successivamente invia l'eventuale valore letto al ModalController. Nel CameraComponent viene definito anche il suo aspetto grafico mostrato nella Figura 4.8.

Nel ChatComponent ho implementato il metodo _initGenericQRCode() il quale aspetta di ricevere il valore letto dal lettore di QR code^[g] che richiama quindi il metodo sendReply() di AzzurraService per dare inizio al il 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^[g], con la differenza che i metodi analoghi a `_openQRcode()`, `_openDatePicker()` e `_openTimePicker()`, non utilizzano il ModalComponent ma l'ion-datetime, una componente grafica offerta da Ionic che può essere configurato per chiedere una data oppure un intervallo temporale; il risultato viene mostrato nella Figura 4.7 e nella Figura ???. Quindi in questi due metodi ho definito come si devono presentare i due *picker* e quindi non c'è stato bisogno di un *component* apposito che li gestisca come per il QRSCANNER.

5.4 Risultati

Nella Figura 5.5 viene mostrata la *chat* tra Azzurra e l'utente per la visualizzazione della pianificazione, sia per un singolo giorno (prima figura) sia per tutta la settimana (seconda figura).



Figura 5.5: Richiesta di visualizzazione della pianificazione

Nella Figura 5.6 viene mostrata la *chat* tra Azzurra e l'utente per la richiesta di visualizzazione del prenotazione dell'utente (prima figura) e la richiesta di scannerizzare il QR code^[g] per usufruire del posto prenotato (seconda figura).

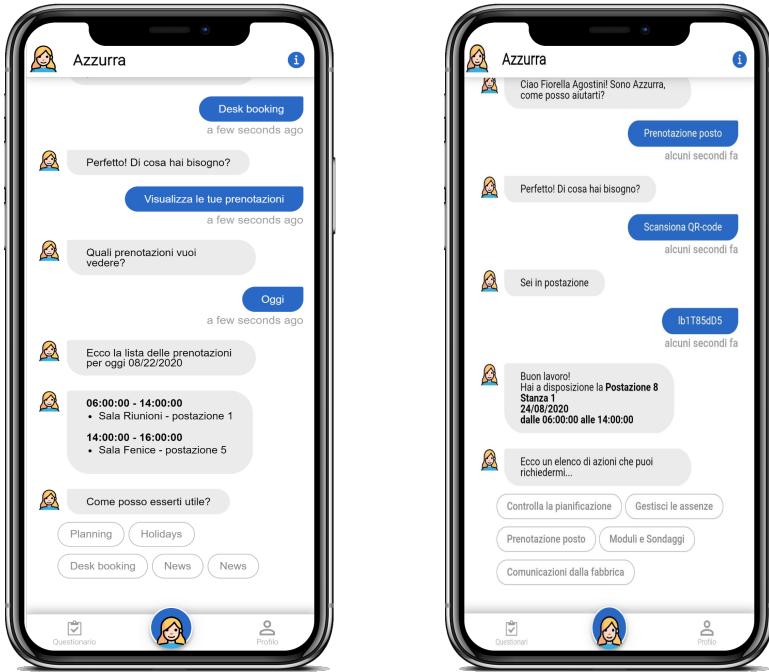


Figura 5.6: Richiesta di visualizzazione delle prenotazioni e scannerizzazione di un QR code



Figura 5.7: Richiesta di inserimento di una nuova prenotazione

Nella Figura 5.6 viene mostrata la *chat* tra Azzurra e l’utente per la richiesta di inserimento di una nuova prenotazione. Viene perciò richiesto il giorno e l’intervallo di tempo in cui fare la prenotazione, in quale stanza deve essere il posto da prenotare e quale posto a sedere vuole prenotare tra quelli disponibili secondo i parametri inseriti.

5.5 Considerazioni

I risultati ottenuti hanno soddisfatto tutti i requisiti stabili e sono stati soddisfacenti per il tutor aziendale. Dai risultati precedentemente elencati si sono fatte delle considerazioni su come potessero essere migliorati in termini di *user experience*. L’architettura^[g] che supporta Azzurra permette di tenere lo storico dei messaggi e lo stato della conversazione per una certa durata, in modo da rendere migliore l’interazione tra utente e Azzurra. Un ulteriore miglioramento che può essere adottato è l’introduzione dei comandi vocali. Al momento l’inserimento dei dati da parte dell’utente avviene principalmente tramite comandi *touch*. Talvolta questa interazione può risultare limitante per alcuni utenti ad esempio con limitazioni alla vista, all’uso delle mani o con il *device* con schermo troppo piccolo. Grazie ai comandi vocali invece queste limitazioni risulterebbero annullate.

Dalla possibile introduzione dei comandi vocali nascono però altre considerazioni. Il riconoscimento vocale deve essere sufficientemente accurato da capire ciò che dice l’utente perché altrimenti può provocare una sensazione negativa di irritazione o frustrazione nell’utente dovuta al fatto che ciò che dice non viene capito dall’applicazione e quindi non offre una buona *user experience*. In sintesi, un possibile modo per migliorare l’*user experience* è l’introduzione dei comandi vocali con un’accurata valutazione a priori di rischi e costi da affrontare.

6 | TESTING

Nel seguente capitolo verranno descritte le tecnologie utilizzate per costruire una test-suite per l'applicazione mobile e verrà esposto il piano di test stabilito con il tutor aziendale ed i risultati ottenuti.

6.1 Test End to End

Il *test End-to-End* è una metodologia di *testing* dell'interfaccia grafica che viene vista dagli utenti dell'applicazione. Ha lo scopo di testare in modo automatizzato, dall'inizio fino alla fine, se tutti flussi di esecuzione dell'applicazione si stanno comportando come progettato, senza che vengano rilevati degli errori che andrebbero a inficiare sulla qualità dell'applicazione stessa. Perciò il *test E2E* consiste nel simulare degli scenari utente reali, ad esempio interazioni attraverso *clic* sui bottoni da parte degli utenti, in modo da verificare che l'applicazione si comporti nel modo corretto garantendo che i requisiti di alto livello del prodotto siano soddisfatti. Il *test E2E* verifica inoltre che l'intera applicazione, su tutte le possibili interazioni dell'utente, trasmetta le informazioni corrette tra le varie componenti garantendo che tutto funzioni come un unico sistema coerente. Un esempio di *test E2E* è la simulazione dei passi che l'utente deve fare per autenticarsi nell'applicazione.

Il *test* quindi può essere così composto:

1. Inserisci l'*username* nella casella di testo;
2. Inserisci la *password* nella casella di testo;
3. Clicca il bottone di *login*;
4. Verifica che ti trovi nella pagina principale dell'applicazione dopo l'autenticazione e non più nella pagina di *login*.

Come scritto, i *test E2E* eseguono l'applicazione proprio come se fosse un utente in un ambiente il più vicino possibile alla realtà. Infatti i *test E2E* dovrebbero avere una comprensione praticamente nulla dei componenti interni dell'applicazione al fine di evitare accoppiamenti non necessari. Ne consegue che i test di unità e di integrazione sono i luoghi in cui dovrebbe essere nota la composizione interna dell'applicazione e quindi dove i mock^[8] possono essere utilizzati per attivare particolari rami di codice che si ha l'esigenza di testare. Il fatto che i *test E2E* non siano a conoscenza della struttura interna e del funzionamento dell'applicazione che stanno testando, li classifica come *test black-box*.

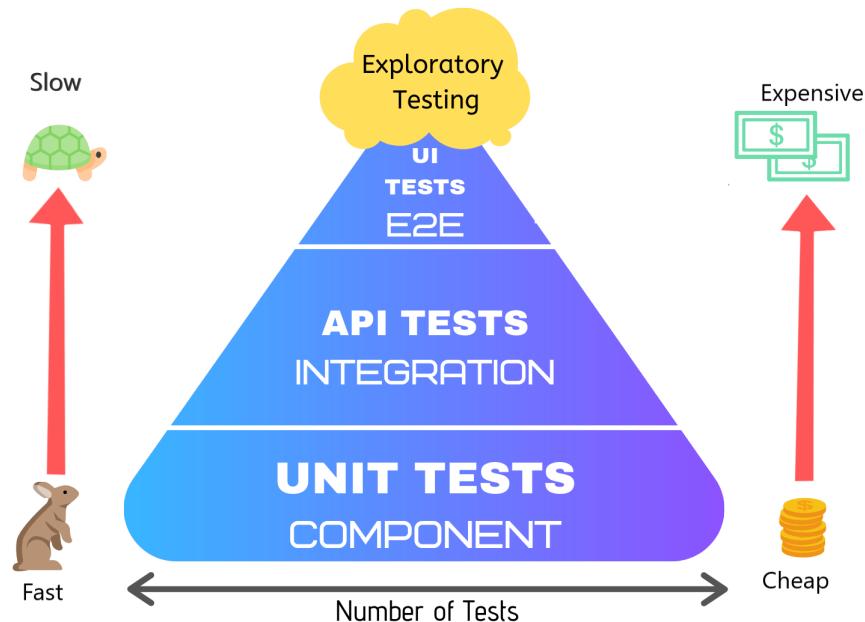


Figura 6.1: Piramide dei test

Come mostra la Figura 6.1, la piramide dei *test* illustra il numero proporzionale di *test* per ogni tipologia in relazione alla loro velocità di esecuzione e ai costi per implementarli. Sebbene i *test* E2E forniscano un grande supporto per verificare la correttezza dell'applicazione, sono significativamente più lenti e più costosi rispetto ai test di unità a causa della loro elevata complessità computazionale. Perciò è fondamentale trovare il giusto equilibrio di *test* E2E da implementare mantenendone i vantaggi e senza aumentare i costi per l'implementazione. Una buona applicazione di *test* E2E può essere ad esempio il *testing* di azioni ripetitive che grazie alla loro automatizzazione, risultano essere meno costosi e più efficaci rispetto ai *test* manuali.

6.2 Tecnologie per il testing

Per implementare i *test* E2E sia per la *dashboard* di AWMS sia per l'applicazione *mobile* si sono usate diverse tecnologie tra loro.

Per il *testing* della *dashboard* si sono usate le seguenti tecnologie:

- * **Selenium WebDrive;**
- * **Protractor;**
- * **Cucumber.**

Selenium è un framework^[g] che permette di automatizzare l'esecuzione dei *test* all'interno dei browser web^[g]. Ciò è possibile grazie a un insieme di API^[g] detto *API WebDriver*. Esse sono uno standard World Wide Web Consortium (W3C) che rappresenta un'interfaccia di controllo remoto capace di manipolare elementi del Document Object Model (DOM)^[g] nelle pagine web e di comandare il comportamento dei programmi utente. Inoltre fornisce un protocollo, noto come JSON Wire Protocol, per il

trasferimento dei dati alle varie piattaforme che permettono di automatizzare i *test* sui browser web^[g] come:

- * GeckoDriver per Mozilla Firefox;
- * ChromeDriver per Google Chrome;
- * SafariDriver per Apple Safari;
- * InternetExplorerDriver per Microsoft Internet Explorer;
- * MicrosoftWebDriver o EdgeDriver per Microsoft Edge;
- * OperaChromiumDriver per Opera.

Selenium per eseguire i *test* utilizza un'architettura client-server. Infatti, Selenium utilizza un server^[g] web in cui espone l'API WebDriver come API Representational State Transfer (REST)^[g] e perciò rimane in ascolto per la ricezione dei comandi che compongono i *test*, sotto forma di richieste HTTP^[g]. Quando arrivano le richieste, le esegue su un browser web^[g] e fornisce una risposta HTTP^[g] che rappresenta il risultato dell'esecuzione del comando. Grazie alla standardizzazione delle API^[g] e all'utilizzo di un architettura client-server, è possibile eseguire *test* scritti in diversi linguaggi purché supportino l'invio di richieste HTTP^[g].

Protractor è un framework^[g] di *test* E2E per applicazioni Angular2+ e AngularJS. Offre un insieme di "localizzatori" cioè metodi che permettono di ricavare gli elementi o il valore degli attributi dall'applicazione web e di simulare dei *click* come se fosse un utente umano. Protractor all'inizio richiede la presenza di un server Selenium in modo da mandare richieste HTTP^[g] per eseguire i *test* E2E.

Infine, Cucumber permette di definire i vari passi che deve fare un *test* automatizzato per simulare le azioni di un utente. Questi passi vengono dichiarati attraverso il linguaggio Gherkin.

Quindi, grazie a Cucumber, vengono dichiarati i cosiddetti *step* ovvero i passi che devono essere fatti all'interno del *test*. L'insieme degli *step* definisce lo scenario dei *test*, ad esempio uno scenario di *test* può essere il processo di autenticazioni e gli *step* l'inserimento dei dati e l'invio. I vari *step* vengono poi implementati in un linguaggio di programmazione, in questo caso TypeScript, utilizzando i metodi offerti da Protractor. Il fatto che vengano dichiarati i passi dei *test* permette a Cucumber di supportare la BDD: quando vengono eseguiti i *test* Cucumber controlla se Selenium sta eseguendo le azioni specificate all'interno di un browser web^[g]. Al termine dell'esecuzione Cucumber stabilisce l'esito per ogni *test* e produce dei *report* sui *test* appena eseguiti documentandone l'esito e la struttura come mostrato in Figura 6.2.

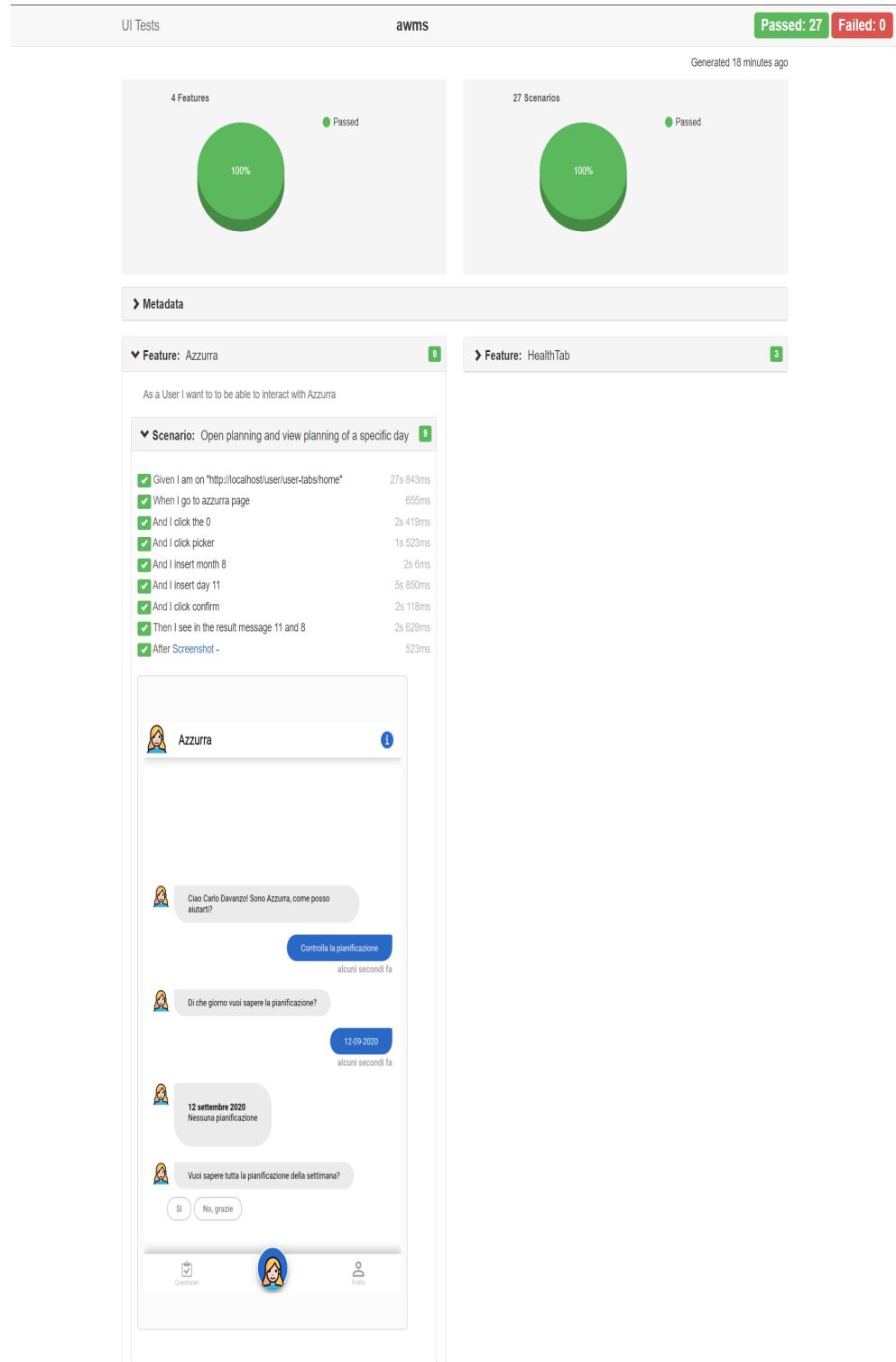


Figura 6.2: Una parte del report dei test per il mobile generato da Cucumber

Per l'applicazione *mobile* si è continuato a utilizzare Protractor e Cucumber mentre al posto di Selenium si è utilizzato Appium perché offre l'automazione dei *test* per applicazioni native^[g], applicazioni web mobile^[g] e applicazioni ibride^[g], con la particolarità di essere multi-piattaforma. Selenium invece permette l'esecuzione di *test* solo su browser web^[g]. Appium utilizza la stessa architettura client-server di Selenium e da esso prende anche l'utilizzo dell'API WebDriver, in particolare utilizza una variante per il *mobile* del protocollo JSON Wire Protocol noto con il nome di Mobile JSON Wire Protocol.

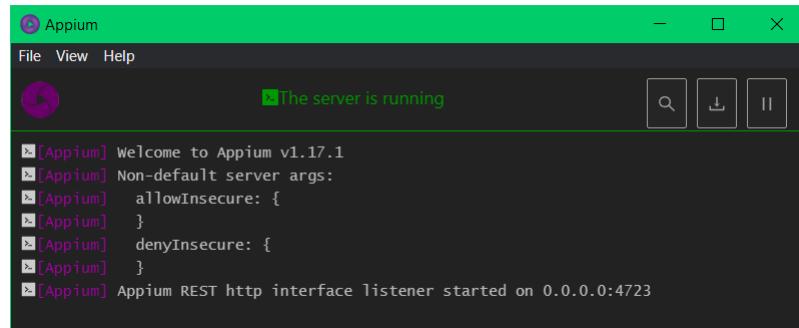
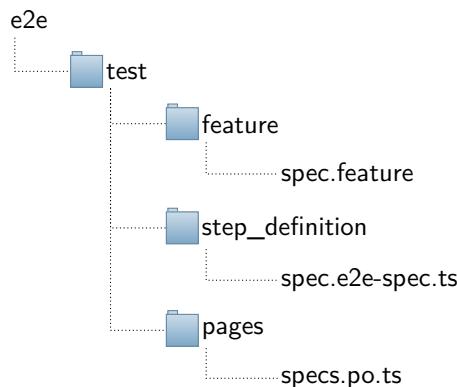


Figura 6.3: Schermata del server Appium

Di conseguenza Appium permette di utilizzare qualsiasi linguaggio o framework^[g] che supporti l'invio di richieste HTTP^[g] senza necessariamente compilare un codice o un framework^[g] specifico di Appium perché, grazie al Mobile JSON Wire Protocol, utilizza i framework^[g] per l'automazione dei *test* come UIAutomator per Android^[g] e XCTest per iOS^[g].

6.3 Convezione

Su suggerimento del tutor aziendale si è strutturata la *test-suite* da me implementata nel seguente modo:



- * **feature:** Contiene i file.feature dove vengono indicati gli scenari e gli *step* dei *test* scritti nell'linguaggio Gherkin. Ogni file.feature contiene *test* di scenari affini;

- * **step_definition:** Contiene i file.e2e-spec.ts dove vengono implementati gli *step*, utilizzando i metodi dei file.po.ts. Esiste un file.e2e-spec.ts per ogni file.feature;
- * **pages:** Contiene le cosiddette PageObject, le PageObject sono un design pattern^[g] che permettono di scrivere *test* più puliti. Infatti, queste PageObject contengono metodi che vengono spesso riutilizzati da più *test* e inoltre, contiene metodi che estraggono gli elementi dalla vista. Adottando questo design pattern^[g] si ha i vantaggi che qualora il modello dell'applicazione cambiasse, sarà sufficiente aggiornare solo la PageObject e in generale non si genera duplicazione del codice.

6.4 Test eseguiti

In comune accordo con il tutor aziendale, sono stati individuati i seguenti *test* E2E per verificare l'intera applicazione *mobile* ovvero Azzurra e le due sezioni Questionario e Profilo. Inoltre sono stati testati anche alcuni aspetti della *dashboard* di AWMS. Per ogni *test* viene assegnato un codice composto dalla lettera **T** e da una seconda lettera che può essere **M** nel caso in cui il *test* sia per l'applicazione *mobile* o la lettera **D** nel caso sia per la *dashboard*. Infine il codice termina con un numero progressivo.

Nella tabella sottostante per ogni *test* viene riportato il suo codice, una descrizione e l'esito del *test*.

Codice	Descrizione	Esito
TM1	Testare che l'utente possa inserire <i>e-mail</i> e <i>password</i> e cliccare il bottone invio per autenticarsi, nel caso i dati siano corretti l'utente deve essere renderizzato nella pagina Questionari.	Superato
TM2	Testare che l'utente possa inserire <i>e-mail</i> , <i>password</i> e cliccare il bottone invio per autenticarsi, nel caso i dati siano uno o tutti errati l'utente deve essere rimanere nella pagina di autenticazione.	Superato
TM3	Testare che nel caso in cui esista una versione più recente rispetto a quella in esecuzione, appaia all'apertura dell'applicazione un messaggio in primo piano che avvisi della nuova versione disponibile.	Superato
TM4	Testare che il messaggio in primo piano che avvisa della nuova versione disponibile, sia interagibile, cioè che si possa cliccare i suoi buttoni.	Superato
TM5	Testare che una volta che l'utente sia autenticato, se è il suo primo accesso, possa chiudere la finestra che contiene la guida per l'utilizzo dell'applicazione, la quale appare al primo accesso che si fa nell'applicazione.	Superato
TM6	Testare nella sezione Questionari, se la prima finestra del Questionario della salute (vedi la prima immagine della Figura 3.4) abbia i vari buttoni dei sintomi tutti cliccabili.	Superato

Tabella 6.1: Tabella del tracciamento dei test E2E

Codice	Descrizione	Esito
TM7	Testare nella sezione Questionari, se la seconda finestra del Questionario della salute (vedi la seconda immagine della Figura 3.4) abbia i vari bottoni sulla salute delle persone che vivono con te, siano tutti cliccabili.	Superato
TM8	Testare nella sezione Questionari, se la terza finestra del Questionario della salute (vedi la terza immagine della Figura 3.4) abbia i vari bottoni dei posti in cui sei stato, siano tutti cliccabili.	Superato
TM9	Testare nella sezione Questionari, che venga visualizzato esito positivo(vedi prima immagine della Figura 3.5) dalla compilazione del questionario se l'indice di contagiosità calcolato è basso.	Superato
TM10	Testare nella sezione Questionari, che venga visualizzato esito negativo(vedi seconda immagine della Figura 3.5) dalla compilazione del questionario se l'indice di contagiosità calcolato è alto.	Superato
TM11	Testare nella sezione profilo che tutti i bottoni presenti siano cliccabili e che aprano la loro corrispondente finestra.	Superato
TM12	Testare nella sezione profilo, nella finestra Informazioni personali, se la ricerca degli indirizzi, per poter cambiare indirizzo di domicilio, produca risultati coerenti con quello che si sta cercando.	Superato
TM13	Testare nella sezione profilo, nella finestra Informazioni, si possa selezionare uno dei risultati della ricerca per il cambio dell'indirizzo di domicilio	Superato
TM14	Testare nella sezione profilo, nella finestra Informazioni, che tutti i bottoni siano cliccabili	Superato
TM15	Testare nella sezione profilo, nella finestra Informazioni, che le modifiche fatte a i dati una volta cliccato il bottone salva, rimangano salvate anche dopo la chiusura della finestra	Superato
TM16	Testare nella sezione profilo, Gestione password, si possa cambiare la <i>password</i> quindi che si possa inserire la <i>password</i> attuale, inserire una nuova <i>password</i> , la sua conferma e cliccare sul bottone salva.	Superato
TM17	Testare nella sezione profilo, che tutte le finestre aperte dopo il <i>click</i> sui bottoni, possano essere chiuse.	Superato
TM18	Testare nella sezione Azzurra, nella funzionalità "Pianificazione" si possa visualizzare la pianificazione in uno specifico giorno.	Superato
TM19	Testare nella sezione Azzurra, nella funzionalità "Pianificazione" si possa visualizzare la pianificazione della settimana corrente.	Superato
TM20	Testare nella sezione Azzurra, nella funzionalità "Gestione assenze" si possa richiedere l'inserimento di una nuova assenza.	Superato

Tabella 6.2: Tabella del tracciamento dei test E2E

Codice	Descrizione	Esito
TM21	Testare nella sezione Azzurra, nella funzionalità "Gestione assenze" si possa visualizzare la lista delle assenze fatte.	Superato
TM22	Testare nella sezione Azzurra, nella funzionalità "Gestione assenze" si possa aprire il calendario dei giorni in cui si può prendere un permesso.	Superato
TM23	Testare nella sezione Azzurra, nella funzionalità "Prenotazione posto" che, si possa visualizzare le prenotazioni dei posti a sedere fatte.	Superato
TM24	Testare nella sezione Azzurra, nella funzionalità "Prenotazione posto" che, si possa inserire una nuova prenotazione di un posto a sedere.	Superato
TM25	Testare nella sezione Azzurra, nella funzionalità "Prenotazione posto" che, si possa aprire lo scanner di QR code ^[g] .	Superato
TM26	Testare nella sezione Azzurra, che si possa aprire il menu delle shortcuts.	Superato
TM27	Testare che siano raggiungibili le sezioni Questionari, Azzurra, Profilo, dopo l'autenticazioni.	Superato
TD28	Testare nella sezione Task che all'inizio non ci sia nessun <i>task</i> selezionato.	Superato
TD29	Testare nella sezione Task che tutti i <i>task</i> siano selezionabili.	Superato
TD30	Testare nella sezione Task che una volta selezionato un <i>task</i> , appaia a lato un menu a tendina.	Superato
TD31	Testare nella sezione Task che la funzionalità di ricerca produca risultati coerenti con quanto si sta cercando.	Superato
TD32	Testare nella sezione Safety che tutti i bottoni siano cliccabili e nel caso in cui aprano delle nuove finestre, queste possano essere chiuse.	Superato
TD33	Testare nella sezione Safety che la funzionalità di ricerca produca risultati coerenti con quanto si sta cercando.	Superato

Tabella 6.3: Tabella del tracciamento dei test E2E

6.5 Considerazioni

I *test* prodotti sono stati in linea con quanto richiesto dal tutor aziendale. Dall'esperienza vissuta sull'implementazione dei *test* E2E si sono fatte delle considerazioni interessanti. All'inizio è stata riscontrata una difficoltà dovuta alla natura asincrona dei *test*: fallivano perché erano più veloci ad eseguirsi rispetto alla renderizzazione delle componenti grafiche che dovevano testare. Un esempio esplicativo è un *test* che non trovava una componente grafica perché non aspettava la sua renderizzazione e falliva anche nonostante la componente grafica fosse correttamente creata. Si è quindi capito quanto fosse oneroso in termini di tempo implementare i vari *test* E2E. Ciononostante, grazie all'utilizzo delle PageObject e dei loro metodi, è stato risparmiato del tempo riutilizzandone alcuni implementati precedentemente.

Da tutto ciò si è compresa la grande onerosità dei *test* E2E, non solo per l'implementazione ma anche per eseguirli. Infatti per eseguire solo ventisette *test* per il *mobile* si è impiegato più di quindici minuti mentre con l'aggiunta dei *test* per la *dashboard* si è arrivati a più di venti minuti.

Infine un aspetto critico rilevato è stato che alcuni rami d'esecuzione non possono essere testati o se ci si prova richiedono troppo tempo per essere testati, ad esempio non è stato possibile verificare in modo completamente automatizzato se lo *scanner* di QR code^[g] fosse in grado di funzionare correttamente perché non si poteva generare un mock^[g] di QR code^[g] da mostrare. Si poteva comunque fare un *test* semi-automatizzato, cioè quando il *test* apriva la fotocamera per scannerizzare, si visualizza manualmente un QR code^[g] in modo che lo *scanner* scansioni e verifichi il corretto funzionamento.

Si considera perciò che i *test* E2E diano un grande valore aggiunto nella qualità del prodotto purchè siano utilizzati per operazioni ripetitive e facilmente implementabili.

7 | CONCLUSIONI

7.1 Consuntivo finale

Rispetto al piano di lavoro esposto nella sezione §2.5, sono state apportate delle modifiche. Ciononostante, il progetto di stage si è concluso soddisfacendo tutti gli obiettivi e i prodotti richiesti e pianificati nonostante le ore totali pianificate inizialmente fossero trecentoventi ed il totale effettivo ammonta a trecentoquattro: tutte le attività pianificate si sono concluse con due giorni d'anticipo previsti nella nona settimana.

Nelle attività della prima settimana c'è stato un aumento di dodici ore perché lo studio approfondito delle tecnologie da utilizzare ha richiesto più tempo del previsto. Per le attività della seconda settimana, su decisione del tutor aziendale, si è spostato l'implementazione dei *test E2E* per il *mobile* alla sesta settimana, ampliando l'attività con il *testing* di tutte le funzionalità dell'applicazione *mobile* e non solo alcune funzionalità come era stato pianificato inizialmente. Per le attività della terza settimana sono state necessarie quattro ore in più a causa di un *bug* manifestatosi in QR Scanner, egregiamente gestito e risolto. Nelle attività della quarta e della quinta settimana si sono risparmiate per ogni settimana otto ore. Tra le attività della sesta settimana si è aggiunto l'implementazione dei *test E2E* per il *mobile* questo perché non è stato possibile effettuare l'implementazione delle notifiche push^[g] a causa di esigenze aziendali cioè, l'azienda aveva l'esigenza di avere già subito implementate le notifiche push^[g], così sono state implementate dai membri dell'azienda. L'attività è stata sostituita dall'attività di documentazione delle notifiche push^[g] oltre all'attività di *testing*. Per le attività della ottava settimana sono state richieste meno ore di quanto pianificato risparmiando così otto ore.

Di seguito viene mostrata la tabella riportante il consuntivo finale del progetto di stage.

Attività	Ore Pianificate	Ore Effettive	Scostamento
Studio delle tecnologie, Angular 2+ e Ionic, da utilizzare durante lo stage	24	36	12
Studio di componenti dell'architettura di sistema di Azzurra, creazione di <i>test</i> per la <i>dashboard</i> di AWMS e per l'applicazione <i>mobile</i>	40	40	0

Tabella 7.1: Tabella riassuntiva del consultivo delle attività per il progetto di stage

Attività	Ore Pianificate	Ore Effettive	Scostamento
Continuazione studio delle componenti del sistema di Azzurra e analisi, progettazione e implementazione di flussi conversazionali.	40	44	+4
Scritture di documentazione per le componenti di Azzurra.	40	32	-8
Continuazione studio di altre componenti di AWMS.	40	32	-8
Documentazione delle componenti AWMS e implementazione notifiche push ^[g] .	40	48	+8
Progettazione, implementazione e documentazione di <i>template engine</i> multilingua.	40	40	0
Studio della gestione dei comportamenti <i>mobile application</i> in condizioni di mancanza di connettività.	40	32	-8
Continuazione ottava settimana.	16	non svolte	-

Tabella 7.2: Tabella riassuntiva del consultivo delle attività per il progetto di stage

7.2 Raggiungimento degli obiettivi

Al termine del progetto di stage sono stati raggiunti tutti gli obiettivi pianificati, validati dalla consegna dei prodotti attesi indicati nella sezione §2.3.

Di seguito vengono riportati gli obiettivi che fanno riferimento alla loro pianificazione descritta nella sezione §2.2.

Obbligatori

- * **OB-1:** Raggiunto. Attraverso l'implementazione dei flussi conversazionali e della *test-suite* per l'applicazione *mobile* e per la *dashboard* di AWMS è stato dimostrato il raggiungimento dell'obiettivo.

Desiderabili

- * **OD-1:** Raggiunto. Durante l'analisi delle componenti utili per l'implementazione dei flussi conversazionali e della *test-suite* per l'applicazione *mobile* e per la

dashboard di AWMS non è stato richiesto particolare aiuto al tutor aziendale lavorando in autonomia;

- * **OD-2:** Raggiunto. Durante la progettazione delle componenti per l'implementazione dei flussi conversazionali e della *test-suite* per l'applicazione *mobile* e per la *dashboard* di AWMS non è stato richiesto particolare aiuto al tutor aziendale lavorando in autonomia.

7.2.1 Riepilogo

Di seguito viene mostrata la tabella riportante il resoconto di tutti gli obiettivi con il relativo stato alla data di fine stage.

Codice	Obiettivo	Esito
OB-1	Competenza nello sviluppo delle singole attività identificate con i linguaggi PHP e Typescript.	Raggiunto.
OD-1	Capacità autonoma di analisi delle singole attività delle soluzioni tecniche viste durante il progetto.	Raggiunto.
OD-2	Capacità autonoma di progettazione delle singole attività delle soluzioni tecniche viste durante il progetto.	Raggiunto.

Tabella 7.3: Tabella riassuntiva degli obiettivi pianificati del progetto di stage

7.3 Consegnna dei prodotti

Al termine del progetto di stage sono stati consegnati tutti i prodotti attesi. Di seguito vengono riportati i prodotti consegnati nella *repository* aziendale su GitLab e sulla piattaforma Confluence.

- * **Analisi tecnica:** È stata consegnata la documentazione dell'analisi tecnica del relativa alle componenti di AWMS e di ciò che è stato prodotto durante lo stage;
- * **Software:** I flussi conversazionali e la *test-suite* che ho prodotto sono stati consegnati nella *repository* aziendale su GitLab e sono pronti per ad essere integrati con i prodotti dell'azienda.

7.3.1 Riepilogo

Di seguito viene mostrata la tabella riportante il resoconto di tutti i prodotti con il relativo stato della consegna alla data di fine stage.

Prodotto	Esito
Descrizione dell'analisi svolta e soluzione identificata, sarà redatta sulla piattaforma documentale aziendale Confluence.	Consegnato.
Implementazione <i>software</i> della soluzione identificata, redatta con l'I-DE di sviluppo identificato per il progetto e depositata sul <i>repository</i> GitLab di riferimento.	Consegnato.

Tabella 7.4: Tabella riassuntiva dei prodotti pianificati del progetto di stage

7.4 Analisi retrospettiva

Dopo essersi concluso il progetto di stage ho analizzato tutto il lavoro svolto prendendo consapevolezza di ciò che è stato fatto e di cosa ho acquisito e imparato da questa nuova esperienza. Inoltre ho compreso l'importanza dell'esperienza lavorativa nel settore della tecnologia che durante il mio percorso di studi non avevo colto.

In seguito verranno riportate le conoscenze e le competenze acquisite, le tecnologie e strumenti utilizzati ed una valutazione personale sull'esperienza di stage.

7.4.1 Conoscenze acquisite

Durante l'esperienza di stage è stato possibile apprendere nuove conoscenze e a raffinare quelle già in possesso. Nello specifico:

- * framework^[g] Angular2+: Angular è stato utilizzato per produrre un applicazione *mobile* composta da una gerarchia di componenti che permettessero la gestione degli eventi sull'interfaccia grafica. Sul framework^[g] Angular avevo delle conoscenze pregresse già prima dell'esperienza di stage ma grazie a quest'ultima ho potuto ampliarle e migliorare l'utilizzo dei metodi offerti, soprattutto per quanto riguarda l'ottimizzazione delle prestazioni;
- * framework^[g] Ionic: Ionic è stato utilizzato per sviluppare l'applicazione *mobile* offrendo componenti grafiche ottimizzate per il *mobile* e permettendo di utilizzare Angular e Cordova insieme. Ho perciò appreso questo nuovo framework^[g] utile per poter sviluppare applicazioni *mobile*;
- * framework^[g] Apache Cordova è stato utilizzato per sviluppare un'applicazione *mobile* con tecnologie web come HTML, CSS e JavaScript e quindi poter sviluppare un'applicazione ibrida^[g] dando accesso anche ai sensori del dispositivo *mobile*.
- * *template-engine*: Ho utilizzato il *template-engine*, nello specifico *Handlebars*, per definire *template* di testi in vari lingue. Grazie all'esperienza di stage ho potuto conoscere il concetto di *template-engine* e di capirne le grandi potenzialità;
- * *test E2E*: I *test E2E* svolti durante il progetto di stage mi hanno permesso di ampliare le mie conoscenze sulla qualità del *software* scoprendo e mettendo in pratica un nuovo tipo di *testing* che prima non conoscevo.

7.4.2 Competenze acquisite

Grazie all'esperienza di stage affrontata, sono maturato dal punto di vista professionale e ho acquisito numerose nuove competenze. Nello specifico:

- * Sviluppo di applicazioni *mobile* ibride: Durante il progetto di stage è stato richiesto di implementare alcune funzionalità vedi §5 per l'applicazione ibrida [g] AWMS Azzurra. Ho quindi imparato ad utilizzare le tecnologie per l'implementazione e compreso i vantaggi dati dallo sviluppo un'applicazione ibrida [g] rispetto a un'applicazione nativa [g];
- * Metodologia di lavoro SCRUM [g]: Nell'esperienza di stage sono entrato in contatto con il framework [g] agile per la gestione dei progetti *software* SCRUM [g] utilizzato dall'azienda che mi ha ospitato. È stato perciò formativo vedere applicato questo modello che avevo studiato in modo teorico nei corsi di Ingegneria del Software e Tecnologie Open-Source.
- * Spirito di imprenditorialità: Durante l'esperienza di stage ho avuto la possibilità di entrare in contatto con l'attività di *business* dell'azienda ospitante. Infatti è incentivato da parte dell'azienda che ogni componente del team esponesse idee su nuove funzionalità da aggiungere ai prodotti già esistenti o idee per nuovi prodotti futuri da sviluppare. Ho perciò potuto capire quali opportunità possano esserci nel mondo del lavoro nell'ambito dell'informatica ma soprattutto in quello della digitalizzazione delle aziende.

7.4.3 Tecnologie e strumenti utilizzati

Nelle attività di stage è stato necessario utilizzare nuove tecnologie e strumenti. Nello specifico si è utilizzato:

- * GitLab: È stato utilizzato per gestire il versionamento dei flussi conversazionali e della *test-suite* sviluppati mantenendo lo storico di tutte le modifiche effettuate;
- * Jira Software: È stato utilizzato per assegnarmi i vari *task* pianificati durante lo stage;
- * Jira Confluence: È stato utilizzato per redigere e consegnare la documentazione richiesta dal tutor aziendale;
- * Ionic Angular e Cordova: Durante lo stage sono stati per sviluppare alcune funzionalità per un'applicazione ibrida [g];
- * Selenium, Protractor, Cucumber e Appium: Sono stati utilizzati per sviluppare i *test* E2E.

7.4.4 Valutazione personale

Valutando questa nuova esperienza fatta, mi ritengo soddisfatto del percorso fatto durante lo stage. Grazie a questa nuova esperienza ho potuto acquisire nuove conoscenze come Ionic e Cordova che mi hanno permesso di apprendere un modo alternativo più veloce e semplice di costruire un'applicazione *mobile* rispetto allo sviluppo con linguaggi nativi. Ho arricchito le mie conoscenze in Angular, soprattutto grazie ai colleghi con cui sono stato affiancato e ho imparato soluzioni migliori a quelle che già conoscevo. Ho inoltre imparato che cosa sono, come implementarli e come funzionano

i *test* E2E.

Infine mi è stata data l'opportunità di capire com'è il mondo del lavoro nell'ambito dell'informatica e di passare dalla teoria alla pratica per quanto riguarda la metodologia di lavoro SCRUM^[8]. Nonostante le restrizioni dovute al COVID-19 ho avuto la fortuna di lavorare in presenza presso l'azienda, rispettando le norme sanitarie negli uffici dell'azienda AzzurroDigitale e trovando un ambiente ospitale e confortevole dove poter lavorare con tranquillità. Durante lo stage sono stato affiancato non solo dal tutor aziendale Carlo Davanzo ma anche dai membri del team a cui Carlo fa capo. Ho perciò lavorato insieme a persone molto disponibili, nonostante i loro impegni lavorativi e molto preparate e simpatiche con cui è stato un piacere lavorare.

Sono infine stato soddisfatto dei risultati ottenuti sui quali ho fatto delle considerazione precedentemente esposte nelle sezioni §5.5 §6.5. Ritengo perciò che il progetto di stage da me sostenuto sia stato molto positivo e istruttivo per i risultati ottenuti, per le conoscenze e competenze acquisite e per i rapporti stretti con il personale dell'azienda che mi ha permesso di effettuare il tirocinio formativo.

ACRONIMI E ABBREVIAZIONI

API *Application Program Interface.* 10, 11, 13, 14, 24, 30, 51, 56, 57, 73, 77

AWMS *Advanced Workforce Management System.* 1–3, 5, 7–9, 13, 16, 19, 23, 25, 30, 40, 56, 60, 65–67, 69

BDD *behavior-driven development.* 11, 57

CSS *Cascading Style Sheets.* 10, 11, 16, 51, 68, 73

DBMS *Database Management System.* 14, 15, 19, 74

DOM *Document Object Model.* 56, 74

E2E *Test End to End.* iii, 5, 11, 55–57, 60, 63, 65, 68–70

GDPR *General Data Protection Regulation.* 18, 75

HTML *Hyper Text Markup Language.* 9–11, 16, 30, 40, 51, 68, 73

HTTP *Hyper Text Transfer Protocol.* 14, 20, 21, 23, 24, 31, 57, 59, 75–77

HTTPS *Hyper Text Transfer Protocol over Secure Socket Layer.* 13, 15, 21, 75

i18n *Internazionalizzazione.* 40

JSON *JavaScript Object Notation.* 19, 25, 30, 34, 36, 40, 51, 76, 77

PHP *Hypertext Preprocessor.* 6, 11, 14, 67

REST *Representational State Transfer.* 57, 77

Sass *Syntactically Awesome StyleSheets.* 10

SDK *Software Development Kit.* 10, 77

URL *Uniform Resource Locator.* 31, 77, 78

W3C *World Wide Web Consortium.* 56, 74

XHTML *eXtensible Hyper Text Markup Language.* 10

XML *eXtensible Markup Language.* 10, 77

GLOSSARIO

Android È un sistema operativo per dispositivi *mobile* sviluppato da Google e basato sul kernel Linux, progettato principalmente per *smartphone* e *tablet*, interfacce utente specializzate per televisori (Android TV), automobili (Android Auto), orologi da polso (Wear OS), occhiali (Google Glass). L'attuale ultima versione è Android 11. 11, 15, 59, 76

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. 71

Applicazione ibrida In informatica si intende applicazioni sviluppate con tecnologie web e vengono eseguite localmente all'interno di un'applicazione nativa. Grazie a ciò possono interagire con il dispositivo ad'esempio utilizzare la fotocamera. 10, 11, 16, 19, 59, 68, 69

Applicazione nativa In informatica si intende un'applicazione scritta e compilata per una specifica piattaforma utilizzando i linguaggi di programmazione e librerie supportati dal particolare sistema operativo *mobile*. 11, 51, 59, 69

Applicazione web mobile In informatica si intende pagine web ottimizzate per dispositivi *mobile* scritte utilizzando tecnologie web, in particolare HTML, JavaScript e CSS. Inoltre, le applicazioni web non possono accedere alle funzionalità del dispositivo ad'esempio la fotocamera. 11, 59

Architettura In informatica con il termine architettura, in questo caso intesa come architettura *software*, è l'organizzazione fondamentale di un sistema, definita dai suoi componenti, dalle relazioni reciproche tra i componenti e con l'ambiente, e i principi che ne governano la progettazione e l'evoluzione. iii, 3, 5, 13, 16, 20, 25, 54

Back-end in informatica con il termine back-end si intende la parte che si occupa di ricevere in input dati inseriti dall'utente e di elaborarli per rispondere alle richieste dell'utente. Dopo l'elaborazione dei dati il back-end produce un risultato che sarà compito del front-end^[g] mostrarlo. 8, 13–16, 19, 22–24, 40, 41, 75

Esecuzione in background In informatica con il termine background, si intende l'esecuzione dei processi di un *software* dove non viene richiesto l'intervento dell'utente, tanto da non essere a lui visibile tale esecuzione. Nel caso di un'applicazione *mobile* significa che nello schermo non viene visualizzata l'applicazione in esecuzione. Resta comunque attiva ma non interagibile con l'utente finché è in background. 24

Base64 In informatica con il termine base64, si intende un sistema di codifica che consente la traduzione di dati binari in stringhe di testo ASCII cioè un insieme di codici per la codifica dei caratteri. I dati vengono rappresentati sulla base di sessantaquattro caratteri ASCII diversi. 21

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. iii, 2, 5, 15, 16, 18, 22, 25, 26, 29–31, 36, 37, 40, 41, 43, 46

Browser web In informatica si intende un'applicazione per l'acquisizione, la presentazione e la navigazione di risorse sul web. Permette la visualizzazione dei contenuti ipertestuali, e la riproduzione di contenuti multimediali. Tra i browser più popolari vi sono Google Chrome, Internet Explorer, Mozilla Firefox, Microsoft Edge, Safari, Opera. 10, 11, 56, 57, 59, 75

Client In informatica con il termine client si intende un entità presente in un rete di comunicazione che accede ai servizi o alle risorse messe a disposizione da un'altra componente detta server^[g], la cui comunicazione tra client e server^[g] è regolata da insieme di regole e norme detti protocolli di comunicazione. Insieme al server^[g] forma l'architettura client/server. 10, 11, 20, 21, 77, 78

Database in informatica con il termine database (ing. base di dati) si intende una collezione di dati ben organizzati e ben strutturati, gestiti in modo integrato da un sistema per la gestione delle basi di dati, costituiscono una base di lavoro per utenti diversi con programmi diversi. I prodotti *software* per la gestione dei database sono indicati con il termine DBMS^[g]. 2, 3, 11, 14, 15, 19, 22, 23, 25, 74, 75

DBMS In informatica con il termine *Database Management System (DBMS)* (ing. sistema di gestione di basi di dati) si intende un sistema *software* progettato per consentire la creazione, la manipolazione e l'interrogazione efficiente di database^[g]. 71

Design pattern In informatica e specialmente nell'ambito dell'Ingegneria del Software con il termine design pattern, si intende di una descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo del *software*, ancor prima della definizione dell'algoritmo risolutivo della parte computazionale. È un approccio spesso efficace nel contenere o ridurre i costi per lo sviluppo del *software*. 60

DOM In informatica con il termine *Document Object Model (DOM)* (ing. modello a oggetti del documento) si intende una forma di rappresentazione dei documenti strutturati in modo gerarchico. È lo standard ufficiale del W3C per la rappresentazione di documenti strutturati in maniera da essere neutrali sia per la lingua che per la piattaforma. 71

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

Firebase È la piattaforma *mobile* di Google che ti aiuta a sviluppare applicazioni *mobile*. Firebase offre tutto ciò che dovrebbe offrire un back-end^[g] quindi, funzionalità di autenticazione, un database^[g] (quello di Firebase è di tipo No-SQL), servizi di *hosting* e algoritmi di machine learning^[g] per l'apprendimento automatico. 15, 24

Esecuzione in foreground In informatica con il termine foreground (ing. primo piano) si intende l'esecuzione dei processi di un *software* dove può essere richiesta l'interazione dell'utente ma che comunque l'utente sa dell'esecuzione di tali processi. Nel caso di un'applicazione *mobile* significa che nello schermo viene visualizzata l'applicazione in esecuzione con la quale l'utente può interagire. 24

Framework In informatica con il termine framework si indica un insieme di elementi *software* che un programmatore può usare o modificare per realizzare un programma. Rappresenta un'astrazione composta da elementi universali e riutilizzabili con lo scopo di facilitare lo sviluppo di un programma e di far applicare buone norme di programmazione. Inoltre, un framework può offrire programmi di supporto, librerie, compilatori e documentazione per l'utilizzo. 10, 11, 13–16, 51, 56, 57, 59, 68, 69

Front-end in informatica con il termine front-end si intende la parte visibile all'utente di un programma e con cui egli può interagire solitamente è un'interfaccia utente. Perciò il front-end è la parte di un sistema *software* che gestisce l'interazione con l'utente, ricevendo da esso un input da cui viene prodotto (dal back-end^[g]) un output da mostrare all'utente. iii, 5, 7, 8, 12–14, 73

GDPR Per *General Data Protection Regulation (GDPR)* (ing. Regolamento generale sulla protezione dei dati) si intende un regolamento dell'Unione europea in materia di trattamento dei dati personali e di *privacy*, adottato il 27 aprile 2016, pubblicato sulla Gazzetta ufficiale dell'Unione europea il 4 maggio 2016 ed entrato in vigore il 24 maggio dello stesso anno ed operativo a partire dal 25 maggio 2018. Il testo affronta anche il tema dell'esportazione di dati personali al di fuori dell'UE e obbliga tutti i titolari del trattamento dei dati (anche con sede legale fuori dall'UE) che trattano dati di residenti nell'UE ad osservare e adempiere agli obblighi previsti.. 71

HTTP In informatica con il termine *Hyper Text Transfer Protocol (HTTP)* (ing. protocollo di trasferimento di un ipertesto) si intende un insieme di regole e norme che regolano la trasmissione e la comunicazione d'informazione nella rete Internet. Questo trasmissione d'informazioni avviene sotto forma di scambi di messaggi tipicamente tra il client che può essere un browser web^[g], e un server. 71

HTTPS In informatica con il termine *Hyper Text Transfer Protocol over Secure Socket Layer (HTTPS)* (ing. protocollo di trasferimento di un ipertesto basato su un strato di sicurezza) si intende un insieme di regole e norme che regolano la trasmissione e la comunicazione d'informazione nella rete Internet in modo sicuro cioè il contenuto della trasmissione non è interpretabile da entità diverse dal mittente o dal/dai destinatario/i. Per la comunicazione viene utilizzato il

protocollo HTTP^[g] all'interno di una connessione criptata dal protocollo *Secure Sockets Layer (SSL)* garantendo così riservatezza dei dati cioè il contenuto della trasmissione è visibile solo al mittente e al destinatario, integrità dei dati cioè il contenuto della trasmessione non viene alterato e autenticazione di comunicazione. 71

iOS È un sistema operativo *mobile* sviluppato da Apple per iPhone, iPod touch e iPad. Le versioni principali di iOS vengono distribuite ogni anno. L'attuale versione, iOS 13, è stata distribuita al pubblico il 19 settembre 2019. 11, 15, 59

JSON In informatica con il termine *JavaScript Object Notation (JSON)*(ing. Notazione degli oggetti JavaScript) si intende un formato testuale standard, usato per rappresentare dati strutturati basati sulla sintassi degli oggetti in JavaScript. È comunemente utilizzato per l'interscambio di dati fra applicazioni client/server. Risulta essere facile da comprendere e da scrivere per le persone mentre per le macchine risulta essere un formato leggero e veloce da analizzare. 71

Licenza MIT La Licenza MIT è una licenza di *software* libero. È una licenza permissiva, cioè permette il riutilizzo nel *software* proprietario sotto la condizione che la licenza sia distribuita con tale *software*. 10

Linguaggio di markup In informatica con il termine linguaggio di markup, si intende un gruppo di regole detti marcatori, attraverso le quali vengono descritti i meccanismi di rappresentazione di un testo. 9

Machine learning Nell'ambito dell'informatica, (ing. apprendimento automatico) l'apprendimento automatico è una variante alla programmazione tradizionale nella quale in una macchina si predispone l'abilità di apprendere qualcosa dai dati in maniera autonoma, senza istruzioni esplicite. 2, 75

Mock In informatica con il termine mock, si intende un oggetto che cerca di riprodurre il comportamento di un oggetto reale in modo controllato, con l'obiettivo di testare il comportamento di altri oggetti reali che dipendono dall'oggetto che si sta simulando con il mock. 55, 63

Notifica push In informatica si intende un tipo di messaggistica istantanea grazie alla quale il messaggio perviene al destinatario senza che questo debba effettuare un'operazione di scaricamento. Tale modalità è quella tipicamente usata da applicazioni come WhatsApp o da servizi di sistemi operativi come Android^[g], oppure da numerose applicazioni derivate da siti web come, ad esempio, il servizio meteo o quello delle notizie. 6, 8, 9, 14, 15, 24, 65, 66

Open-source in informatica con il termine open-source (ing. sorgente libero) si intende un *software* per cui chi lo ha sviluppato rinuncia alla proprietà del *software* dando libero accesso a tutto il codice sorgente a chiunque, e quindi è permesso a tutti di contribuire nello sviluppo del codice al fine di migliorarlo, aggiungere nuove funzionalità o correggere errori all'interno del codice. 10–12

Plant manager È colui (ing. responsabile di stabilimento) 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. Raccoglie e analizza i dati di produzione per trovare eventuali spazi di miglioramento. Si occupa della sicurezza dei lavoratori e quella degli impianti inoltre, monitora le apparecchiature di produzione e, in caso di necessità, della loro riparazione o sostituzione. 2, 13, 14, 23

Pooling In informatica con il termine pooling si intende una procedura attraverso la quale periodicamente viene eseguita una operazione. Nel caso delle comunicazioni tra client^[g] e server^[g] il pooling è la richiesta periodica del client^[g] di dati al server^[g] per controllare se i dati che ha sono aggiornati. 20

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*. ix, 3, 18, 33, 38, 43–46, 48, 49, 51, 52, 62, 63

REST In informatica con il termine *Representational State Transfer (REST)* (ing. trasferimento di stato rappresentativo) si intende un approccio architettonico alla creazione di web API^[g] basato sul protocollo di comunicazione HTTP^[g]. Viene imposto che le API^[g] devono permettere di accedere a delle risorse attraverso un URL^[g], utilizzare il formato JSON e XML, non avere uno stato cioè non deve essere memorizzato cioè che è stato fatto e infine, utilizzare i metodi del HTTP^[g], GET, POST, PUT, DELETE. 71

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 degli 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, 69, 70

SDK in informatica con il termine *Software Development Kit (SDK)* (ing. pacchetto di sviluppo per *software*) si intende una collezione di strumenti per lo sviluppo *software* contenuti all'interno di un singolo pacchetto installabile all'interno del proprio sistema. Tutto ciò viene offerto per facilitare la creazione di applicazioni. Questi strumenti solitamente sono specifici per il particolare tipo di *hardware*, sistema operativo e linguaggio di programmazione utilizzati per lo sviluppo *software*. 71

Server In informatica con il termine client si intende un entità presente in un rete di comunicazione che offre dei servizi o dalle risorse a un'altri componenti presenti nella rete detti client^[g], la cui comunicazione tra client^[g] e server è regolata da insieme di regole e norme detti protocolli di comunicazione. Insieme al client^[g] forma l'architettura client/server. 10, 20, 21, 57, 74, 77, 78

URL In informatica con il termine *Uniform Resource Locator (URL)* (ing. localizzare di risorse uniformi) si intende una sequenza di caratteri che identifica univocamente l'indirizzo di una risorsa su una rete di computer, come ad esempio un documento, un'immagine, un video, tipicamente presente su un server^[g] e reso accessibile a un client^[g]. 71

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, 15, 18, 20, 21, 30, 36

BIBLIOGRAFIA

Siti web consultati

Agile manifesto. URL: <https://agilemanifesto.org/iso/it/manifesto.html>.

Angular2+. URL: <https://angular.io/>.

Appium. URL: <http://appium.io/>.

Confluence. URL: <https://www.atlassian.com/it/software/confluence>.

Cordova. URL: <https://cordova.apache.org/>.

Cucumber. URL: <https://cucumber.io/>.

Gitlab. URL: <https://about.gitlab.com/>.

Handlebars. URL: <https://handlebarsjs.com/guide/>.

Ionic. URL: <https://ionicframework.com/>.

Jira Software. URL: <https://www.atlassian.com/it/software/jira>.

Linguaggio CSS. URL: <https://www.w3schools.com/css/>.

Linguaggio HTML. URL: <https://www.w3schools.com/html/>.

Linguaggio TypeScript. URL: <https://www.typescriptlang.org/>.

Protractor. URL: <https://www.protractortest.org/#/>.

Selenium. URL: <https://www.selenium.dev/documentation/en/>.

WebStorm. URL: <https://www.jetbrains.com/webstorm/>.