

# Università Politecnica delle Marche

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica e dell'Automazione

---



**Tesi di Laurea**

**Progettazione e implementazione di un'app Android e di un sistema di back-end per la gestione delle ordinazioni in una pizzeria**

**Design and implementation of an Android app and a back-end system for the management of the orders in a pizzeria**

Relatore

Prof. Domenico Ursino

Candidato

Marco La Gala

---

**Anno Accademico 2018-2019**



---

# Indice

<b>Introduzione</b> .....	3
<b>1 La programmazione mobile</b> .....	7
1.1 La nascita di un nuovo strumento: il telefono cellulare .....	7
1.1.1 I primi telefoni cellulari .....	7
1.1.2 9 Gennaio 2007: il primo iPhone .....	8
1.1.3 Google ad Android: cambio di rotta .....	10
1.2 Il rafforzamento dello <i>strumento</i> : la nascita delle app .....	11
1.2.1 Che cosa è un'applicazione? .....	11
1.2.2 Il cambio di paradigma: la svolta degli App Store .....	12
1.2.3 Accoglienza del pubblico .....	12
1.2.4 Nascita di una nuova figura: lo sviluppatore mobile .....	13
1.3 Le app odierne .....	14
1.3.1 App native .....	15
1.3.2 Le web app .....	15
1.3.3 App ibride .....	16
<b>2 L'ecosistema Android</b> .....	17
2.1 Perché Android? .....	17
2.1.1 La nascita di Android .....	17
2.1.2 Ecosistema .....	19
2.1.3 Diffusione delle versioni .....	19
2.2 Architettura di Android .....	20
2.2.1 Applications .....	21
2.2.2 Application Framework .....	22
2.2.3 Native Libraries .....	23
2.2.4 Android Runtime .....	24
2.2.5 Linux Kernel .....	24
2.3 Strumenti per gli sviluppatori .....	24
2.3.1 Java Development Kit .....	24
2.3.2 Scelta di un IDE: Android Studio .....	25
2.3.3 Android SDK .....	26
2.4 Le componenti di un'applicazione Android .....	26

## IV **Indice**

2.4.1	Le activity e il loro ciclo di vita . . . . .	26
2.4.2	I Service . . . . .	27
2.4.3	I BroadcastReceiver . . . . .	27
2.4.4	I ContentProvider . . . . .	28
2.4.5	Il Manifest . . . . .	28
2.4.6	Le Risorse . . . . .	28
<b>3</b>	<b>Analisi dei requisiti . . . . .</b>	<b>31</b>
3.1	Descrizione del progetto . . . . .	31
3.2	Requisiti funzionali e non funzionali . . . . .	32
3.2.1	Requisiti funzionali . . . . .	32
3.2.2	Requisiti non funzionali . . . . .	32
3.3	Attori e casi d'uso . . . . .	33
3.3.1	Diagramma dei casi d'uso . . . . .	33
3.3.2	Tabella dei casi d'uso . . . . .	33
3.4	Matrice di mapping . . . . .	35
<b>4</b>	<b>Progettazione . . . . .</b>	<b>37</b>
4.1	Struttura dell'applicazione . . . . .	37
4.1.1	Mappa dell'applicazione . . . . .	37
4.1.2	Mockup . . . . .	38
4.1.3	Flow Chart o Diagrammi di Flusso . . . . .	39
4.2	Progettazione della base di dati . . . . .	44
4.2.1	Progettazione concettuale . . . . .	45
4.2.2	Progettazione logica . . . . .	49
<b>5</b>	<b>Implementazione . . . . .</b>	<b>55</b>
5.1	Le funzionalità dell'applicazione . . . . .	55
5.2	Componenti di Firebase utilizzate . . . . .	58
5.2.1	Firebase Storage . . . . .	61
5.2.2	Firebase Database . . . . .	63
5.2.3	Firebase Authentication . . . . .	64
5.3	Implementazione delle componenti dell'applicazione . . . . .	66
5.3.1	Blocco dell'app con Shared Preferences . . . . .	66
5.3.2	Sistema fidelizzazione utenti . . . . .	67
5.3.3	Sessione Utente . . . . .	68
<b>6</b>	<b>Discussione in merito al lavoro svolto . . . . .</b>	<b>71</b>
6.1	Punti di forza del sistema realizzato . . . . .	71
6.2	Punti di debolezza del sistema realizzato . . . . .	72
6.3	Analogie con altri sistemi simili esistenti sul mercato . . . . .	73
6.4	Differenze rispetto ad altri sistemi simili esistenti sul mercato . . . . .	74
<b>7</b>	<b>Conclusioni . . . . .</b>	<b>77</b>
	<b>Ringraziamenti . . . . .</b>	<b>79</b>
	<b>Riferimenti bibliografici . . . . .</b>	<b>81</b>

---

## Elenco delle figure

1.1	Martin Cooper con il <i>Motorola DynaTAC 8000X</i> . . . . .	7
1.2	<i>IBM Simon Personal Communicator</i> . . . . .	8
1.3	<i>Nokia 3310</i> . . . . .	9
1.4	I telefoni più usati nel 2007 . . . . .	9
1.5	Il primo modello di iPhone . . . . .	10
1.6	Android prima e dopo l'iPhone . . . . .	11
1.7	<i>T-mobile G1</i> : il primo telefono Android . . . . .	11
1.8	I due market più famosi oggi: App Store e Google Play . . . . .	12
1.9	Numero di download e di applicazioni sull'App Store . . . . .	13
1.10	Ricavi medi dei primi 100 sviluppatori di app mobile negli anni . . . . .	14
1.11	Le varie tipologie di app . . . . .	14
1.12	Le tre piattaforme su cui sviluppare app native . . . . .	15
1.13	Le web app sui vari dispositivi . . . . .	16
1.14	I framework più famosi per sviluppare app ibride . . . . .	16
2.1	Andy Rubin, uno dei fondatori di Android, che indica il logo della sua creazione . . . . .	18
2.2	Ecosistema Google per i clienti business . . . . .	20
2.3	Grafico a torta illustrativo delle distribuzioni Android; dati aggiornati al 7 maggio 2019 . . . . .	21
2.4	Tabella riassuntiva delle distribuzioni Android con la percentuale della distribuzione; dati aggiornati al 7 maggio 2019 . . . . .	21
2.5	I cinque livelli dell'architettura Android . . . . .	22
2.6	I vari passaggi per la compilazione del codice sorgente . . . . .	24
2.7	Architettura Android . . . . .	25
2.8	Ciclo di vita di un'Activity . . . . .	27
2.9	Cartella <code>res</code> all'interno di un file di progetto . . . . .	29
3.1	Elenco degli attori . . . . .	34
3.2	Diagramma dei casi d'uso relativo all'utente non registrato . . . . .	34
3.3	Diagramma dei casi d'uso relativo all'utente registrato . . . . .	35
3.4	Diagramma dei casi d'uso relativo al Cameriere/Staff del ristorante . . . . .	35
3.5	Matrice di mapping relativa all'applicazione . . . . .	36

## VI Elenco delle figure

4.1	Mappa relativa all'applicazione mobile	38
4.2	Mockup relativo alla scheda "Imposta Tavolo"	39
4.3	Mockup relativo alla scheda "Home"	40
4.4	Mockup relativo alla scheda "Ordina"	40
4.5	Mockup relativo alla scheda "Profilo"	40
4.6	Mockup relativo alla scheda "Pietanza"	40
4.7	Mockup relativo alla scheda "Aggiunte"	40
4.8	Mockup relativo alla scheda "Note"	40
4.9	Mockup relativo alla scheda "Carrello"	41
4.10	Mockup relativo alla scheda "Comande"	41
4.11	Mockup relativo alla scheda "Pagamento"	41
4.12	Mockup relativo alla scheda "Login Staff"	41
4.13	Mockup relativo alla scheda "Login Utente"	42
4.14	Mockup relativo alla scheda "Registrazione"	42
4.15	Mockup relativo alla scheda "Visualizza Informazioni"	42
4.16	Mockup relativo alla scheda "Area Riservata"	42
4.17	Mockup relativo alla scheda "Pagamento" per un utente registrato	43
4.18	Mockup relativo alla scheda "Profilo" per un utente registrato	43
4.19	Diagramma di attività relativo alla scelta del piatto da aggiungere al carrello	44
4.20	Diagramma di attività relativo alla richiesta del pagamento degli ordini	45
4.21	Diagramma di attività relativo alla funzionalità del cambio numero del tavolo	50
4.22	Entità principali dell'applicativo sviluppato	51
4.23	Diagramma E/R relativo all'applicativo sviluppato	51
4.24	Dizionario delle entità relativo alla base di dati dell'applicazione	52
4.25	Dizionario delle relazioni relativo alla base di dati dell'applicazione	52
4.26	Elenco degli identificatori principali relativo alla base di dati dell'applicazione	53
4.27	Glossario dei termini relativo alla base di dati dell'applicazione	53
4.28	Traduzione verso il modello relazionale relativo alla base di dati dell'applicazione	54
5.1	Implementazione relativa alla pagina "Imposta Tavolo"	55
5.2	Realizzazione della pagina "Home"	56
5.3	Realizzazione della pagina "Ordina"	56
5.4	Realizzazione della pagina "Profilo"	56
5.5	Realizzazione della pagina "Pietanza"	57
5.6	Realizzazione della pagina "Aggiunte"	57
5.7	Realizzazione della pagina "Note"	57
5.8	Realizzazione della pagina "Carrello"	58
5.9	Realizzazione della pagina "Comande"	58
5.10	Realizzazione della pagina "Pagamento"	58
5.11	Realizzazione della pagina "Login Staff"	58
5.12	Realizzazione della pagina "Login Utente"	59
5.13	Realizzazione della pagina "Registrazione"	59

5.14	Realizzazione della pagina “Visualizza Informazioni”	59
5.15	Realizzazione della pagina “Area Riservata”	59
5.16	Realizzazione della pagina “Impostazioni Staff”	59
5.17	Pop-up che permette di impostare il pagamento del cliente.	59
5.18	Realizzazione della pagina “Pagamento” con utente registrato	60
5.19	Realizzazione della pagina “Profilo” con utente registrato	60
5.20	Elenco dei servizi e degli strumenti messi a disposizione da Firebase	61
5.21	Schematizzazione dell’architettura tradizionale e dell’architettura con Firebase	62
5.22	Realtime Database di Firebase	63
5.23	Esempio di nodo contenente le informazioni di un oggetto di tipo menù	64
6.1	Versione della scheda “Profilo” nelle due lingue disponibili	72
6.2	Esempio di piattaforma web dove il personale può modificare il menù	73
6.3	Sezione relativa al menù dell’app <i>iMenu</i>	74
6.4	Scheda relativa ad un prodotto dell’app <i>Nu-Menu</i>	75
6.5	Sezione relativa alla recensione del servizio dell’applicazione <i>Nu-Menu</i>	76
6.6	Funzionalità del codice QR nell’applicazione <i>Wmenu</i>	76





---

## Elenco dei listati

5.1	Definizione del metodo <code>uploadImageToFirebaseStorage</code> della classe <code>SignInActivity</code> .....	62
5.2	Definizione parziale della classe <code>FirebaseDatabaseHelper</code> .....	63
5.3	Definizione parziale della classe <code>LogInActivity</code> .....	65
5.4	Definizione del metodo <code>AppBlock</code> della classe <code>MainActivity</code> .....	66
5.5	Definizione del metodo <code>Finisci</code> .....	67
5.6	Definizione del metodo <code>onBackPressed</code> della classe <code>Login</code> .....	67
5.7	Definizione del metodo <code>onDataChange</code> della classe <code>ProcediAlPagamento</code> per l'attribuzione dei punti .....	68
5.8	Definizione del metodo <code>NuovaSessione</code> della classe <code>ImpostaTavolo</code> ..	68
5.9	Definizione del metodo <code>collectSessionNumbers</code> della classe <code>ImpostaTavolo</code> .....	69



---

## Introduzione

Negli ultimi anni, in Italia, seppur in ritardo rispetto al resto del mondo, si è visto moltiplicare l'uso di dispositivi mobili come strumenti che permettono di supportare e semplificare molteplici attività. Oggigiorno, infatti, gli utenti installano nel proprio cellulare applicazioni, le quali aiutano e semplificano le attività quotidiane. Queste hanno la caratteristica di essere talmente immediate e a portata di mano, che hanno soppiantato un'ampia gamma di oggetti, partendo dall'agenda, sostituita dall'app del calendario, oppure dalle lettere, rimpiazzate dalle chat di messaggistica, fino a rivoluzionare anche istituti imponenti e duraturi, come le banche, con l'*home banking*.

Uno dei campi in cui, ultimamente, l'utilizzo di questi dispositivi porta notevoli vantaggi è quello della ristorazione. I cellulari, infatti, vengono ampiamente utilizzati durante il processo di ordinazione, in quanto, prima di tutto, permettono di formattare l'ordine secondo uno standard ben preciso, eliminando tutti gli errori di comprensione derivanti dalla scrittura su carta e, in secondo luogo, consentono di inviare contemporaneamente l'ordine a tutte le postazioni fisse, come quelle in cucina o alla cassa, e ai restanti cellulari utilizzati dal personale. Inoltre, permettono, con una moderata spesa iniziale, dovuta all'acquisto dei dispositivi mobile, di ridurre il personale in sala e, di conseguenza, nel lungo periodo, possono garantire ampi margini di guadagno al gestore dell'attività. Proprio per questi motivi, da anni, ormai, i camerieri non utilizzano più le tipiche comande di carta, ma smartphone con applicazioni che comunicano direttamente alla cucina.

Lo step successivo della diffusione di questi dispositivi consiste nel farli utilizzare direttamente dalla clientela. Da qualche anno, infatti, la catena di fast food più nota al mondo, *McDonalds*, ha apportato delle modifiche nei suoi ristoranti posizionando all'ingresso delle postazioni fisse che permettono all'utente di ordinare direttamente i piatti presenti nel menù. Ciò ha portato notevoli vantaggi alla nota catena; infatti le lunghe file presenti alle casse sono notevolmente diminuite: il personale si occupa solo di far pagare il cliente nel caso in cui quest'ultimo abbia scelto il pagamento tramite contanti. In questo modo, lo staff del ristorante, alleggerito dal carico delle ordinazioni, avrà più tempo per fornire un servizio migliore. Esistono, inoltre, notevoli vantaggi nell'utilizzo di queste postazioni da parte della clientela. Esse, infatti, rispetto ai comuni menù cartacei, permettono di scegliere le pietanze in tutta tranquillità; inoltre, il cliente potrà disporre di un menù ricco di immagini,

di informazioni e diviso comodamente in sezioni.

Un ulteriore vantaggio consiste nel fatto che il cliente può decidere come personalizzare il proprio piatto, rimuovendo ingredienti o aggiungendone altrettanti e potrà, in ogni momento della sua ordinazione, visionare il totale dell'ordine o il prezzo del singolo piatto. In questo modo, il ristorante sarà in grado di accontentare anche i clienti più impegnativi e, soprattutto, quelli con esigenze alimentari particolari, come intolleranze o allergie.

Vista la crescita esponenziale nell'utilizzo di questi dispositivi, sempre più attività di ristorazione, e non solo importanti catene, hanno iniziato a comprendere i benefici di questa soluzione, nonostante l'esistenza di un problema che frena l'adozione, ossia lo spaesamento dei clienti più anziani che, sicuramente, con il passare di qualche anno, svanirà.

Il progetto di questa tesi mira allo sviluppo di un'applicazione mobile che permetta al cliente di ordinare autonomamente al ristorante, senza la necessità di un cameriere. Il nostro scopo è fornire un'alternativa valida, e meno dispendiosa in termini di risorse, al sistema di ordinazione classico. Tale alternativa si trasforma in un risparmio di tempo per il cliente, mentre, per il proprietario, comporta un risparmio del personale.

Si è deciso, quindi, di sviluppare un'applicazione da installare nei dispositivi posti sui tavoli del ristorante. Il motivo di questa scelta è dettato dal fatto che, a differenza delle colonnine dette in precedenza, che non si addicono ad un ristorante italiano, il dispositivo mobile permette di raggiungere un buon compromesso tra i costi e i benefici, in quanto il prezzo della singola unità è all'incirca di un centinaio di euro e, inoltre, ha come seconda funzione quella di intrattenere la clientela con i giochi presenti all'interno.

La scelta di utilizzare dispositivi mobili ci permette di associare, a ciascuno di essi, all'inizio del servizio, un particolare tavolo, il quale, poi, potrà essere liberamente modificato, se durante la serata i camerieri ne avessero la necessità (ad esempio, perché cambiano la disposizione dei tavoli). Quindi, si può affermare che la soluzione adottata risulta flessibile in base alle esigenze del ristoratore.

Il gestore del ristorante avrà una spesa iniziale derivante dall'acquisto dei telefoni cellulari necessari, ma questa sarà recuperabile grazie alla riduzione del personale, il quale avrà solo il compito di soddisfare gli ordini. Inoltre, la soluzione proposta permette di eliminare le spese relative ai futuri cambiamenti del menù e consente di effettuare modifiche in modo istantaneo.

Abbiamo deciso, vista l'eterogeneità degli utenti che si interfaceranno con l'app, di renderla quanto più semplice e comprensibile possibile, e di fornire loro la possibilità di richiedere aiuto nel caso in cui avessero qualche problema o incomprensione durante l'utilizzo.

Il compito più importante e difficile di un ristorante è, sicuramente, quello di aumentare i propri clienti, attradone di nuovi e fidelizzando i restanti. Per questo motivo, si è deciso di creare una sezione in cui l'utente potrà accedere ed ottenere sconti e promozioni. Il cliente, guadagnando dei punti ad ogni ordine effettuato, sarà maggiormente invogliato a ritornare al ristorante, in quanto, al successivo pasto, avrà uno sconto particolare.

In conclusione, le funzionalità dell'applicazione, saranno principalmente tre. La prima è quella di permettere al cliente di ottenere informazioni sul ristorante e sui

piatti del menù; la seconda è quella di permettere al cliente di effettuare ordinazioni; la terza è quella di consentirgli di guadagnare dei punti per scontare gli ordini successivi.

Nella presente tesi, dopo una spiegazione generale riguardo il mondo della progettazione mobile e dell'ecosistema Android, sarà illustrata una panoramica generale sull'applicazione sviluppata, soffermandosi, in particolare, sull'analisi dei requisiti, sulla progettazione della base di dati e sull'implementazione delle funzionalità più rilevanti.

La tesi è strutturata nei seguenti capitoli:

- Nel primo capitolo, *La programmazione mobile*, verrà proposta una panoramica sull'evoluzione dei telefoni cellulari in relazione al mondo delle applicazioni e alla figura dello sviluppatore mobile.
- Nel secondo capitolo, *L'ecosistema Android*, tratteremo del sistema operativo del “robottino verde”, concentrandoci sull'architettura e sugli strumenti indispensabili per la programmazione.
- Nel terzo capitolo, *Analisi dei requisiti*, effettueremo, inizialmente, una descrizione generale del progetto, per poi analizzare sia i requisiti funzionali che quelli non funzionali del nostro applicativo, nonché i vari casi d'uso e gli attori che utilizzeranno la nostra applicazione.
- Nel quarto capitolo, *Progettazione*, verrà descritta in dettaglio la progettazione dell'interfaccia dell'applicativo, tramite *Mockup* e *Flow Chart*, e della base dei dati, tramite la progettazione concettuale e quella logica.
- Nel quinto capitolo, *Implementazione*, tratteremo le funzionalità principali dell'applicazione, per poi illustrare l'implementazione delle componenti principali dell'applicativo.
- Nel sesto capitolo, *Discussione in merito al lavoro svolto*, affronteremo un'analisi critica sul progetto sviluppato, considerando sia i punti di forza sia i punti di debolezza dell'applicativo, e confrontandolo con i sistemi esistenti in commercio.
- Nel settimo capitolo, *Conclusioni*, ripercorreremo in sintesi gli argomenti trattati nella tesi e delineremo alcuni possibili sviluppi futuri dell'applicativo.



## La programmazione mobile

*In questo capitolo ripercorreremo la nascita e l'evoluzione dei telefoni cellulari, soffermandoci sul cambiamento dalla tastiera fisica al touchscreen, che permise la diffusione capillare delle app, fino ad arrivare ad esporre le varie tipologie di app che un programmatore odierno può realizzare.*

### 1.1 La nascita di un nuovo strumento: il telefono cellulare

#### 1.1.1 I primi telefoni cellulari

Il 3 aprile 1973, Martin Cooper, allora direttore della sezione *Ricerca e Sviluppo della Motorola*, effettuò la prima telefonata con un nuovo prodotto tecnologico: il telefono cellulare (Figura 1.1). Il prototipo garantiva 30 minuti di conversazione a fronte di 10 ore di ricarica, pesava 1,1 kg ed era considerato “portatile” nonostante le sue dimensioni (228mm x 27mm x 44,4mm). Prima di esso, infatti, Motorola, come altre aziende, installavano i telefoni esclusivamente su veicoli come automobili o treni in quanto, oltre ad essere troppo grandi e pesanti, consumavano talmente tanta energia da poter essere alimentati solo dal motore del veicolo stesso.



**Figura 1.1.** Martin Cooper con il *Motorola DynaTAC 8000X*

Dopo ben dieci anni, nel 1983, Motorola riuscì ad ottenere l'approvazione della *Federal Communications* e, quindi, a commercializzare il *Motorola DynaTAC 8000X*, successore del prototipo presentato nel 1973, al costo di circa \$4.000 (circa \$10.000 oggi). Anche se destinato ad un pubblico di imprenditori o a gente ricca, ottenne un discreto successo principalmente perché permetteva ai suoi utilizzatori di essere sempre raggiungibili.

Se fino ad ora si è parlato esclusivamente di telefono cellulare è perché solo nell'anno 1992 si vide la nascita di un primo *smartphone*: l'*IBM Simon* (Figura 1.2) progettato dall'*IBM* e commercializzato l'anno successivo dalla BellSouth, il quale permetteva, attraverso uno schermo touch previsto di pennino, di inviare e-mail, gestire la rubrica, l'orologio e giocare anche con un videogioco precaricato. Con il termine "smartphone" infatti, secondo il dizionario *Treccani*, si intende un dispositivo che

«unisce alle caratteristiche di un telefono cellulare le potenzialità di un piccolo computer, grazie alla presenza di un sistema operativo completo e autonomo. Tra le funzionalità [troviamo] l'accesso a Internet, la ricezione e l'invio di e-mail, l'elaborazione di file e il servizio GPS.»



**Figura 1.2.** *IBM Simon Personal Communicator*

Tra il 1990 e il 1995, con l'avanzamento del progresso tecnologico e con la riduzione di prezzo e dimensioni, i telefoni cellulari si diffondono tra i consumatori medi fino a diventare la norma, piuttosto che l'eccezione, all'inizio del 2000. Si può ricordare, infatti, il famoso *Nokia 3310* (Figura 1.3), annunciato da *Nokia* il 1° settembre del 2000 e prodotto a partire da ottobre dello stesso anno, il quale è stato uno dei cellulari di maggior successo della storia, avendo raggiunto 126 milioni di esemplari venduti fino al suo ritiro dal mercato nel 2005.

### 1.1.2 9 Gennaio 2007: il primo iPhone

Il 9 gennaio 2007, Steve Jobs sul palco del *Macworld Conference* a San Francisco, presentò il primo telefono *Apple* della storia (Figura 1.5). Iniziò la sua presentazione affermando che oggi avrebbe introdotto tre prodotti rivoluzionari: un nuovo





**Figura 1.3.** *Nokia 3310*

*iPod* con controllo touch, un telefono e un nuovo *device* per navigare in rete. Solo successivamente rivelò che queste caratteristiche erano tutte contenute in un unico oggetto: l'*iPhone*.

Nel 2007, infatti, l'interazione dell'utente con i dispositivi mobili era legata principalmente alla tastiera e, se i dispositivi disponevano anche di uno schermo touch, questo veniva controllato esclusivamente con un pennino (Figura 1.4). Riguardando il video della presentazione, si può quantificare la ventata di novità che introdusse quel dispositivo dalla reazione del pubblico in sala. Lo smartphone della *Apple*, differenziandosi dai principali dispositivi in commercio, eliminava la tastiera fisica per sostituirla con uno schermo maggiorato in dimensione e permetteva di interagire con esso usando semplicemente le dita. Steve Jobs giustificò le sue scelte con la frase «Button and controls can't change», in quanto i tasti della tastiera potevano essere esclusivamente pensati in fase di progettazione e non potevano cambiare né in futuro né in base al tipo di applicazione che si utilizzava perché erano esclusivamente fisici.



**Figura 1.4.** I telefoni più usati nel 2007

Un altro aspetto che si può notare, durante tutta la presentazione, è il modo in cui Steve Jobs cercava di convincere gli utenti che il suo prodotto era semplice da usare «easy to use», minimale, ma, al contempo, in grado di svolgere funzioni complesse. Mostrò al pubblico, infatti, come, con pochi click, si riusciva a recuperare il numero di telefono di uno *Starbucks* dalle mappe per poi chiamarlo ed ordinare del caffè, oppure che per scorrere una lista di canzoni non c'era più bisogno di pigiare un pulsante, ma bastava fare un solo gesto che oggi chiamiamo, comunemente, *swipe up*.

Sei mesi dopo, il 28 giugno 2007, il prodotto arrivò sul mercato al prezzo di 499 dollari e, grazie alla pubblicità e alla ventata di novità, 6 consumatori su 10 già conoscevano il prodotto prima del lancio.



Figura 1.5. Il primo modello di iPhone

### 1.1.3 Google ad Android: cambio di rotta

Anche *Google* in quegli anni stava sviluppando un sistema operativo per dispositivi mobili grazie alle competenze e alle tecnologie ottenute dopo aver comprato, nel 2005, *Android Inc.* per 50 milioni di dollari. La società aveva pianificato di presentarlo nell'estate del 2007, ma l'avvento dell'iPhone qualche mese prima obbligò al team di rivedere completamente il progetto. Avevano, infatti, ideato Android tenendo presente che i dispositivi sui quali sarebbe stato caricato erano sulla falsariga del BlackBerry, lo smartphone più venduto in quegli anni, come è possibile vedere dall'immagine di un prototipo, mostrate in Figura 1.6, rivelate durante la causa *Google v. Oracle America* riguardo l'utilizzo di *Java*.

Volevano, quindi, immettere sul mercato un sistema operativo sviluppato esclusivamente per dispositivi dotati di tastiera e sprovvisti di touchscreen, i quali sarebbero risultati già obsoleti in confronto al nuovo dispositivo Apple. Per questo motivo modificarono le specifiche fissate per Android affermando che il touchscreen sarebbe stato supportato, ma, comunque, rimasero fedeli alla convinzione che questo non avrebbe mai potuto sostituire i tasti fisici.

Il primo telefono con Android, infatti, fu presentato il 23 settembre 2008 con il nome di *T-Mobile G1* (o *HTC dream*), prodotto dalla società taiwanese *HTC* e



Figura 1.6. Android prima e dopo l'iPhone

commercializzato dal carrier telefonico *T-Mobile* (Figura 1.7). Esso era provvisto di uno schermo touch ma comunque, per scrivere, si doveva aprire il telefono ed utilizzare una tastiera fisica. L'avvento di una tastiera software arrivò solo con la Versione 1.5 di Android nel 2009.



Figura 1.7. *T-mobile G1*: il primo telefono Android

## 1.2 Il rafforzamento dello *strumento*: la nascita delle app

### 1.2.1 Che cosa è un'applicazione?

Con il termine *applicazione* (conosciuta anche con l'abbreviazione app) si intende un'applicazione software installabile nei dispositivi mobili, come smartphone o tablet, la quale permette di eseguire un compito ben preciso dell'utente. Un'app è

composta da un'interfaccia grafica propria e riesce a sfruttare le funzionalità del dispositivo e del sistema operativo in cui è installata. Di solito la dimensione di un'app è di qualche MB, in quanto è sviluppata per garantire un'esperienza semplice e limitata.

Infatti, diversamente da un'applicazione desktop la quale viene utilizzata in poche ma lunghe sessioni durante la giornata, un'applicazione mobile viene utilizzata per pochissimo tempo, di solito qualche minuto, ma molto frequentemente. Inoltre, bisogna considerare che gli smartphone ed i tablet sono dispositivi con una potenza ed una capacità di memorizzazione limitate, per cui le app devono risultare leggere e non pesare troppo sulle prestazioni dello smartphone.

### 1.2.2 Il cambio di paradigma: la svolta degli App Store

Alla nascita dei telefoni cellulari le uniche app presenti erano quelle preinstallate dai produttori, come calendario, orologio, e-mail, etc., mentre oggi siamo abituati ad utilizzare molteplici app realizzate da aziende esterne o sviluppatori singoli.

Inizialmente Steve Jobs, durante lo sviluppo dell'iPhone, decise di non permettere agli sviluppatori di programmare app native in quanto voleva che creassero web app per il browser proprietario *Safari*, con gli strumenti già disponibili sul mercato come *Web 2.0* e *Ajax*, senza la necessità di un *SDK (Software Development Kit)* apposito.

Purtroppo, le web app non decollarono mai su iPhone e Jobs cambiò idea, anche esortato dal responso del pubblico, annunciando, ad Ottobre 2007, che un *SDK* per iPhone sarebbe stato disponibile l'anno successivo. Fu così che l'11 Luglio 2008 con l'aggiornamento ad iPhone OS2 fu presentato l'**App Store**, un negozio virtuale di applicazioni che, al lancio, ne contava già 500 disponibili. Queste potevano essere sia gratuite sia a pagamento e venivano divise in categorie per facilitare la ricerca dell'utente.

Oggi il termine *app store* o *market* è diventato di uso comune ed è ampiamente utilizzato anche da altri produttori, per indicare il luogo dove poter trovare le applicazioni da scaricare (Figura 1.8).

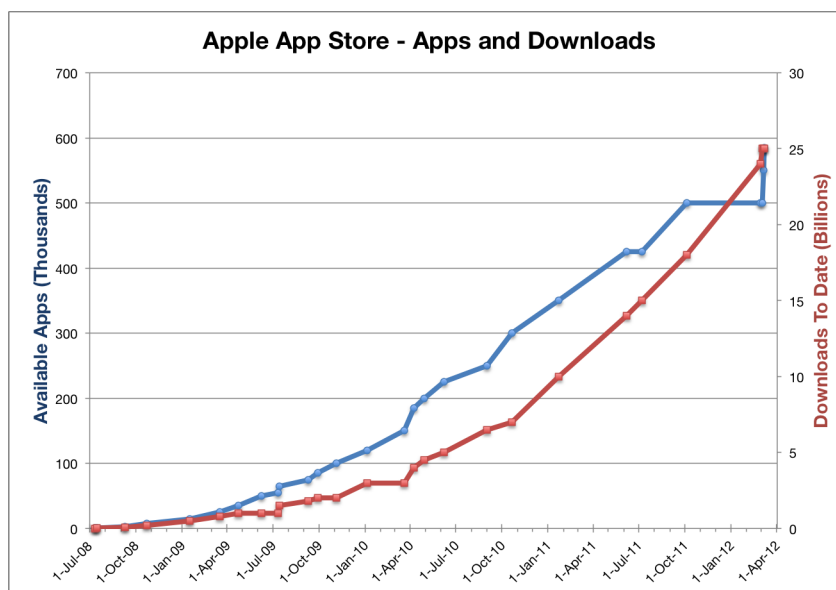


Figura 1.8. I due market più famosi oggi: App Store e Google Play

### 1.2.3 Accoglienza del pubblico

La diffusione delle app negli ultimi 10 anni ha subito una forte crescita. Basti vedere il grafico di Figura 1.9 per riconoscere un andamento esponenziale, sia per quanto riguarda il numero di download degli utenti, sia per il numero di app presenti nello store. Le app, infatti, sono entrate a tutti gli effetti nella nostra routine quotidiana

ed ormai per ogni tipo di attività possiamo scaricarne una. Il motivo principale per cui sono esplose è che, rispetto alle solite applicazioni presenti all'interno del dispositivo sviluppate dalla casa madre, queste possono essere prodotte da una comunità di sviluppatori, la quale è variegata e formata da molte più persone. Gli utenti, in questo modo, hanno ottenuto l'accesso a un "mondo" di funzionalità da installare permettendo, così, di rendere il proprio dispositivo sempre più personale e adatto alle loro esigenze.



**Figura 1.9.** Numero di download e di applicazioni sull'App Store

#### 1.2.4 Nascita di una nuova figura: lo sviluppatore mobile

Con la nascita dei vari store e la diffusione capillare delle app, cresce la figura dello sviluppatore mobile, sia freelance sia operante per un'azienda. Sviluppare un'applicazione mobile è un lavoro a tempo pieno e, per questo motivo, può essere affrontato solo se remunerativo. Gli sviluppatori, infatti, si sono trovati ad avere una vetrina (*App Store* o *Play Store*) dove poter pubblicare la propria applicazione gratis oppure a pagamento. Questa grande base di utenti ha permesso di ottenere ricavi consistenti che hanno contribuito, a loro volta, alla crescita delle app. Si può vedere dal grafico di Figura 1.10, prodotto da *SensorTower*, che i ricavi dei due store aumentano di anno in anno nonostante l'esistenza di una notevole disparità tra i due tipi di market.

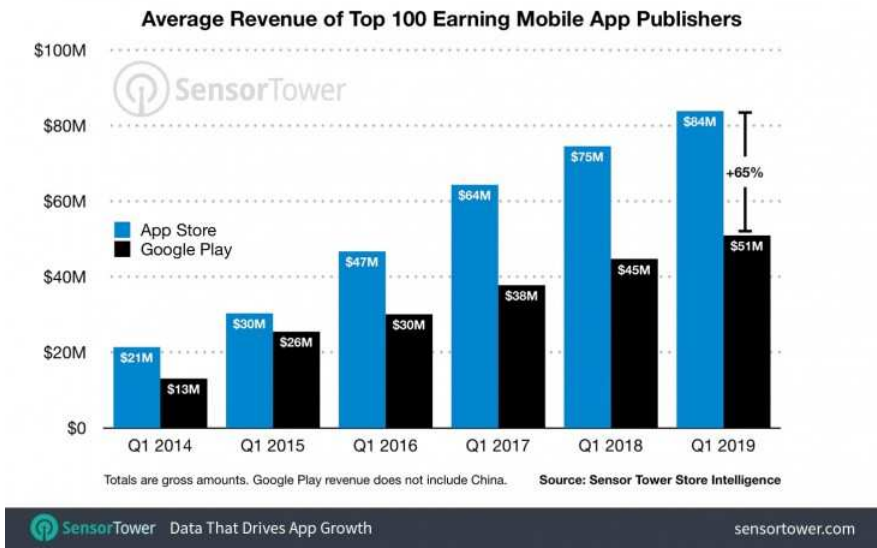


Figura 1.10. Ricavi medi dei primi 100 sviluppatori di app mobile negli anni

### 1.3 Le app odierne

Oggiuno sviluppatore si trova ad avere varie alternative per la realizzazione di un'applicazione mobile (Figura 1.11). Queste possono essere raggruppate in tre macro-categorie: *app native*, *app ibride* e *web app*. La scelta della tipologia giusta dipende dalle esigenze, dal target di riferimento ma, soprattutto, dalle risorse economiche e dal tempo a disposizione. Per questo motivo non sempre una soluzione è migliore delle altre, ma è bene conoscere tutte le possibili alternative, con i loro vantaggi e svantaggi, per effettuare una scelta consapevole.

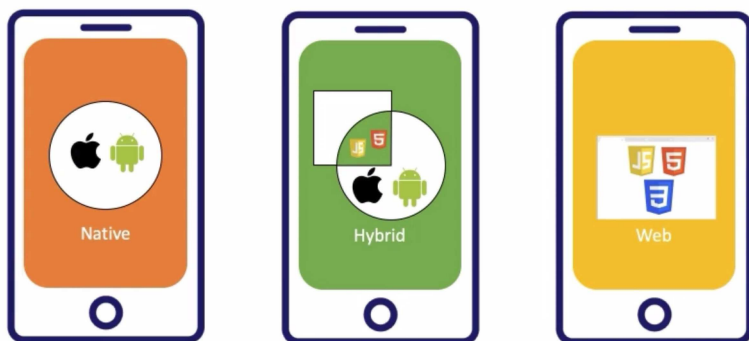


Figura 1.11. Le varie tipologie di app

### 1.3.1 App native

Le app native rappresentano una tipologia di applicazioni che vengono sviluppate su misura per una piattaforma. Per esempio, per iOS sono sviluppate in *Object-C* o *Swift*, mentre per Android in *Java* o *Kotlin*. Di conseguenza, visto che sono specifiche per il sistema operativo di riferimento, esse si adattano perfettamente alla piattaforma per la quale sono state create e possono sfruttare moltissime funzionalità del dispositivo, come la fotocamera, il GPS, la localizzazione o le notifiche push, le quali permettono di inviare avvisi agli utenti, per esempio a fini promozionali (Figura 1.12).

Inoltre, esse risultano più veloci ed affidabili delle altre tipologie di app e permettono di ottenere un'applicazione più reattiva, al che si tramuta in una esperienza utente migliore per l'utilizzatore. Un'app nativa, infine, può anche essere utilizzata in modalità offline, a differenza delle web app, e, quindi, l'utente non deve avere bisogno di una connessione ad Internet costante.

Essere legati ad una piattaforma specifica, però, comporta anche degli svantaggi: se gli sviluppatori vogliono distribuire la loro app su più sistemi operativi, devono programmarla per ognuna di queste piattaforme, con conseguenti costi aggiuntivi, in termini di tempo e denaro, rispetto ad una web app.



Figura 1.12. Le tre piattaforme su cui sviluppare app native

### 1.3.2 Le web app

Le web app sono applicazioni che funzionano come un sito web e, quindi, riescono a lavorare indipendentemente dal tipo di sistema operativo (Figura 1.13). Sono, infatti, sviluppate con i comuni linguaggi di programmazione dei siti web, come *HTML*, *CSS* e *Javascript*. Inoltre, non hanno bisogno di alcuna installazione, visto che sono caricate da un web server ed eseguite direttamente sul browser. Queste app, di conseguenza, non andranno ad inficiare la memoria del telefono, ma, al contempo, hanno bisogno di una connessione Internet per funzionare.

Gli svantaggi principali di queste tipologie di app riguardano il fatto che non consentono performance all'altezza delle app native e, al contempo, sono leggermente più lente perché devono comunicare con la rete. Inoltre, non avendo una propria applicazione sull'*app store* ufficiale del sistema operativo, può risultare più difficile incrementare il numero di utenti che la utilizzano; d'altro canto, di conseguenza, si

ha il vantaggio che le modifiche al codice non hanno bisogno di essere approvate per esser pubblicate e, quindi, sono repentine.



**Figura 1.13.** Le web app sui vari dispositivi

### 1.3.3 App ibride

L'anello mancante tra le due tipologie di app appena descritte sono le app ibride, chiamate anche multiplatforma. Rispetto alle app native sono più rapide da realizzare e meno dispendiose in termini economici; inoltre, viene prodotta una sola versione, indipendentemente dal numero di piattaforme sulle quali si desidera essere presenti. Infine, le app ibride rappresentano ottime soluzioni *web-based*, che si integrano perfettamente con l'hardware e il software dei vari dispositivi e, soprattutto per i costi ridotti, sono l'alternativa migliore per chi vuole affacciarsi al mondo delle app, ma non ha grandi budget da investire.

Tuttavia, tutti questi benefici hanno un costo: le performance di queste app sono inferiori rispetto a quelle delle app native, dal momento che non sono legate alla piattaforma.

Esistono in commercio moltissimi framework che permettono di sviluppare app ibride, come *Xamarin* della Microsoft, *React Native* e *Ionic*, i quali forniscono degli strumenti per semplificare lo sviluppo multiplatforma (Figura 1.14).



**Figura 1.14.** I framework più famosi per sviluppare app ibride



## L'ecosistema Android

*In questo capitolo daremo uno sguardo ravvicinato al sistema operativo Android, soffermandoci in breve sulla sua storia e inoltrandoci, anche, sulla sua architettura. Illustreremo, poi, gli strumenti necessari ad uno sviluppatore per la programmazione mobile spiegando anche i vari componenti che potranno essere utilizzati per la progettazione di un'applicazione.*

### 2.1 Perché Android?

#### 2.1.1 La nascita di Android

Nell'Ottobre del 2003, molto prima che la parola “smartphone” entrasse nell'uso comune ed alcuni anni prima della presentazione dell'iPhone, venne fondata da Rich Miner, Nick Sears, Chris White, e Andy Rubin, una compagnia chiamata *Android Inc* in Palo Alto, California (Figura 2.1).

Il progetto che aveva intenzione di sviluppare la società rimase segreto per molti anni. Una delle poche dichiarazioni ufficiali venne effettuata nel 2003 da Andy Rubin, uno dei fondatori, spiegando che le intenzioni della società erano quelle di sviluppare dispositivi mobile “intelligenti” che avrebbero utilizzato la posizione e le preferenze del proprietario: «smarter mobile devices that are more aware of its owner's location and preferences».

Oggi sappiamo che il progetto originale della società era quello di migliorare il sistema operativo delle fotocamere digitali. Ben presto, però, i fondatori realizzarono che quel mercato non era grande abbastanza per garantire la sopravvivenza dell'azienda ed, inoltre, era in calo da diversi anni, per cui la società decise di rendere Android un sistema operativo portatile che sarebbe andato in competizione con quelli più diffusi all'epoca: *Symbian* e *Microsoft Windows Mobile*.

Nel Luglio del 2005 Andy Rubin presentò il progetto a Lerry Page, fondatore di Google, cercando di convincerlo di scegliere Android come sistema operativo per i futuri dispositivi di Google. Page, però, anche se molto interessato all'idea, sapeva bene, conoscendo ciò che era successo ad *IBM* con la *Microsoft*, che sponsorizzare una società terza significava dargli troppo potere, così decise di acquistare *Android Inc.* per circa 50 milioni di dollari.

Lo scopo dell'acquisizione rimase segreto fino al 2007 quando Google, con un comunicato stampa, affermò che trentaquattro aziende come *Texas Instruments*, *Intel*, *T-Mobile* e *Sprint Nextel* si erano unite insieme a lei per costruire un'interfaccia wireless basata su software open source Linux. Il gruppo si soprannominò *Open Handset Alliance*.

I principali competitor sul mercato, come Microsoft e Nokia, inizialmente, giudicando Android come «un'altra piattaforma Linux», non compresero ciò che Google aveva intenzione di ideare e, allo stesso tempo, non presero precauzioni contro le possibili ripercussioni che si sarebbero potute verificare nei loro confronti.

Altri invece, anche se aspettavano un dispositivo sviluppato direttamente da Google che potesse rivaleggiare contro iPhone, capirono subito la ventata di novità che questo approccio portava. In quegli anni, anche se venivano utilizzati più di 2 miliardi di telefoni cellulari nel mondo ed il numero era in crescente aumento, non c'era un'organizzazione condivisa tra le varie aziende che producevano telefoni cellulari perché ognuna aveva una piattaforma con standard e specifiche proprie. Con l'introduzione di Android le carte in tavola cambiarono, perché questo sistema operativo permise di definire uno standard ed un unico sistema che aziende come HTC, Motorola, LG, e tante altre, potevano utilizzare per i propri telefoni cellulari i quali però, a loro volta, potevano essere personalizzati dai produttori secondo le loro linee guida e con le loro applicazioni proprietarie.

In questo modo ci furono vantaggi sia per le compagnie, le quali potevano concentrare i loro sforzi sull'hardware, sia per gli sviluppatori, i quali potevano rilasciare applicazioni per più dispositivi concentrandosi quindi sull'ampliamento delle funzionalità offerte, sia per i consumatori che compravano un sistema operativo robusto, duraturo nel tempo e dotato di moltissime applicazioni.



**Figura 2.1.** Andy Rubin, uno dei fondatori di Android, che indica il logo della sua creazione

È naturale chiedersi per quale motivo si è deciso di sviluppare un'applicazione scegliendo come sistema operativo Android e non altri, come, per esempio, iOS. Le ragioni che hanno portato alla sua scelta per la realizzazione di questo progetto sono molteplici.

La prima risiede nel fatto che Google “sposa” una filosofia *open source* per lo sviluppo di Android rilasciandolo tramite licenza *Apache*. Questa licenza permette

di usare, distribuire e modificare il software a proprio piacimento ed, inoltre, non vincola le versioni modificate ad avere la stessa licenza.

Google, infatti, dopo aver completato lo sviluppo delle funzionalità del suo sistema operativo, rilascia il codice sorgente attraverso un progetto chiamato *Android Open Source Project* (AOSP), il quale fornisce le informazioni ed il codice necessario alle aziende per creare varianti di Android personalizzate per i loro dispositivi.

Inoltre, il secondo motivo risiede nel fatto che Android, grazie anche alla caratteristica di essere open source, rappresenta oggi il sistema operativo più diffuso al mondo in quanto si stima abbia una fetta di mercato dell'86% con una crescita sempre in positivo negli anni (secondo *IDC: International Data Corporation*). Possiamo riportare questo risultato e capire, di conseguenza, l'importanza del "robotto verde", considerando la diffusione del secondo sistema operativo iOS, ferma solo a circa il 13%.

### 2.1.2 Ecosistema

Come già ribadito in precedenza, Android, negli ultimi anni, si sta diffondendo in molteplici campi anche molto distanti da quello per cui è nato, ossia lo smartphone, spinto anche dalla sua natura open source.

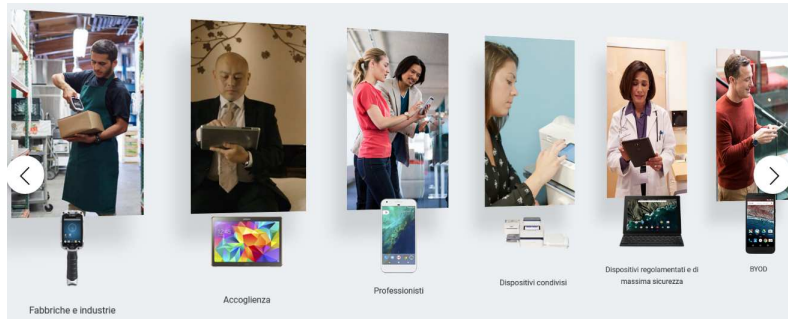
Si può vedere l'introduzione di *Android Auto* nel Marzo del 2015, sistema operativo che permette di mostrare informazioni al guidatore direttamente sul display della macchina, oppure i vari prototipi dei *Google Glass*, occhiali *smart* che permettono, ad esempio, di mostrare all'utente messaggi o mappe direttamente sulle lenti. Negli ultimi anni, inoltre, vediamo che Android si riesce ad adattare e acquista una fetta di mercato consistente anche in dispositivi recenti come gli *smartwatch* (*Wear OS*), ossia piccoli dispositivi con le sembianze di un orologio da tenere al polso, oppure le *Smart tv*, televisori che possono accedere, tramite applicazioni, ai contenuti fruibili online, come video on demand, multimedialità e servizi di streaming (*Android TV*).

Infine, si può vedere dalla Figura 2.2, la quale rappresenta una porzione del sito Android dedicato ai clienti business, come Google stia cercando di far entrare il suo sistema operativo all'interno del mondo business fornendo supporto alle aziende attraverso dispositivi portatili, tablet, oppure dispositivi *rugged*, per finalità specifiche.

In un immediato futuro vedremo sicuramente questo sistema operativo anche nei vari oggetti smart che saranno presenti nelle nostre case in quanto Google, come molti altri competitor, sta investendo molto nell'*Internet of Things* e nella domotica, la cosiddetta "casa intelligente".

### 2.1.3 Diffusione delle versioni

Dopo almeno due versioni *alfa* interne chiamate *Astro Boy* e *Bender*, la versione *beta* e il suo *SDK* vennero rilasciate da Google nel Novembre del 2007. L'*SDK* (*Software Development Kit*) era composto principalmente dagli strumenti di sviluppo, le librerie, un emulatore del dispositivo e la documentazione in inglese correlata da vari esempi di progetto e permise agli sviluppatori di iniziare a creare applicazioni per Android.



**Figura 2.2.** Ecosistema Google per i clienti business

Il 23 Settembre del 2008 venne presentata la Versione 1.0 del sistema operativo, nella quale erano presenti moltissime funzionalità all'avanguardia, come il market per scaricare le applicazioni, il *client* di posta *Gmail* sviluppato da Google, un web browser ed un media player. Solo con la Versione 1.5 del 2009 denominata, anche, *Cupcake*, si iniziarono a definire le varie versioni di Android con nomi di dolci procedendo in ordine alfabetico. Tra il 2009 e il 2010 furono rilasciate ben 5 versioni (*Cupcake*, *Donut*, *Eclair*, *Froyo* e *Gingerbread*) e questo segna il fatto che Android stava riscuotendo un enorme successo e lo sviluppo era spinto al massimo.

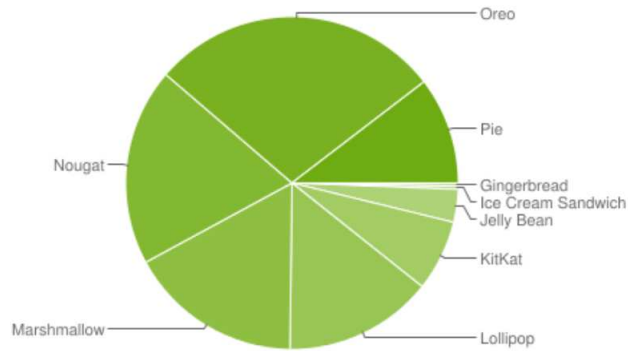
L'ultima versione, uscita nel 2019, interrompe questa denominazione; infatti viene chiamata esclusivamente Android 10 forse anche a causa della difficoltà di trovare un nome adatto che inizi con la lettera dell'alfabeto *Q*.

Android, però, non è esente da *problemi*. Come è possibile notare, anche, dal grafico a torta (Figura 2.3 e 2.4) è un sistema operativo frammentato; troviamo, infatti, che ben 4 versioni precedenti alla *Pie* (Versione 9) sono tuttora utilizzate da moltissimi dispositivi. Quindi moltissimi utenti hanno nel loro telefono un software non aggiornato da quattro anni. Questo è causato principalmente dal fatto che, dopo l'uscita di una nuova versione, i produttori devono modificarla e adattarla con le funzionalità dell'azienda, e non sempre ciò rappresenta una loro priorità causando, quindi, una frammentazione maggiore. Negli ultimi anni Google, per risolvere questo problema, ha avviato un progetto chiamato *Android One* il quale permette ai costruttori di fornire il telefono esclusivamente con la versione stock del sistema operativo, permettendo, così, aggiornamenti più veloci e telefoni più sicuri.

## 2.2 Architettura di Android

Android, come è possibile vedere nella Figura 2.5, non è un semplice sistema operativo ma uno *stack software* formato principalmente da cinque livelli con all'interno diversi componenti. Ogni livello sfrutta le funzionalità di quello sottostante e fornisce funzionalità a quello superiore.

1. Applications
2. Application Framework
3. Native Libraries



**Figura 2.3.** Grafico a torta illustrativo delle distribuzioni Android; dati aggiornati al 7 maggio 2019

Version	Codename	API	Distribution
2.3.3-2.3.7	Gingerbread	10	0.3%
4.0.3-4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%

**Figura 2.4.** Tabella riassuntiva delle distribuzioni Android con la percentuale della distribuzione; dati aggiornati al 7 maggio 2019

4. Android Runtime
5. Linux Kernel

### 2.2.1 Applications

Nel livello più elevato troviamo le *applications* chiamate, in italiano, anche *app di sistema*. Queste costituiscono un insieme di applicazioni fondamentali per l'utilizzo del telefono nel quotidiano, come browser, client e-mail, messaggistica tramite SMS, dialer e app di terze parti. Sono scritte principalmente in *Java* o *Kotlin* compilate,



**Figura 2.5.** I cinque livelli dell'architettura Android

poi, con l'ausilio dell'Android SDK, il quale permette di unire tutte le risorse di un *package* in un file con estensione `apk`.

Queste applicazioni vengono poi distribuite attraverso il Play Store, ossia il market di Google, ma è possibile scaricarle anche dal browser oppure installarle direttamente con il PC tramite il cavo USB.

Una volta installate, il sistema operativo considera ogni applicazione come se fosse un utente differente, assegnando ad essa un ID Utente, e associa i suoi file e le sue risorse a quell'ID in modo tale che altre applicazioni non possano accedervi ed, eventualmente, danneggiarla.

Inoltre, ogni applicazione viene eseguita solo sul processo Linux assegnato ad essa, e ogni processo ha la caratteristica di avere una propria macchina virtuale in modo tale che le applicazioni girino indipendentemente dalle altre.

Infine, per creare un ambiente ulteriormente sicuro, ogni applicazione non può accedere a nessuna altra porzione del sistema senza permesso. Queste però, possono richiedere all'utente, o in fase di installazione o nel momento del bisogno, il permesso per accedere a determinate funzionalità, come contatti, fotocamera, memoria interna etc. Esistono 4 tipi di componenti dell'applicazione che tratteremo meglio nella Sezione 2.4; esse sono *Activities*, *Services*, *Broadcast Receivers* e *Content Providers*.

## 2.2.2 Application Framework

Il secondo livello è l'*Application Framework*; questo raggruppa un insieme di *API* scritte in Java e altre componenti ben specifiche, le quali permettono l'esecuzione di funzionalità fondamentali per la corretta esecuzione delle applicazioni.

Questo livello è quello più importante per gli sviluppatori mobile in quanto contiene tutti gli strumenti e le funzionalità utilizzabili per sviluppare un'app.

Le principali componenti di questo *framework* sono:

- *Activity Manager*, il quale permette di organizzare le varie schermate di un'applicazione in uno stack a seconda dell'ordine di visualizzazione delle stesse sullo schermo dei diversi dispositivi. L'*Activity Manager* è lo strumento fondamentale attraverso il quale l'utente interagisce e naviga nell'applicazione e permette, di

conseguenza, di ripercorrere indietro le varie schermate visualizzate con il tasto *back* dello smartphone.

- *Content Provider*, il quale ha la responsabilità di gestire la condivisione di informazioni tra i vari processi di una stessa app, oppure tra diverse applicazioni.
- *Telephony Manager*, il quale fornisce agli sviluppatori dei metodi per utilizzare le funzionalità caratteristiche di un telefono, come le chiamate, e permette, anche, di rispondere a determinati eventi, come, per esempio, la perdita di rete telefonica.
- *Location Manager*, il quale mette a disposizione API che permettono all'applicazione di utilizzare il GPS (*Global Positioning System*) e, di conseguenza, consentono il corretto funzionamento delle mappe.
- *Resource Manager*, il quale è un componente che ha il compito di gestire le risorse, come grafici e immagini, mettendo a disposizione una serie di API.
- *View System*, il quale gestisce la renderizzazione dei componenti, come pulsanti e blocchi di testo, e degli eventi associati ad essi. Infatti, l'interfaccia grafica di un'applicazione per Android è composta da specializzazioni della classe *View*, ciascuna caratterizzata da una particolare forma e da un diverso modo di interagire con gli elementi dell'app.
- *Notification Manager*, il quale rende disponibili un insieme di strumenti che l'applicazione può utilizzare per inviare una particolare notifica al dispositivo, quest'ultimo dovrà presentarle all'utente sotto forma di piccolo messaggio a scomparsa.
- *Package Manager*, il quale gestisce il processo d'installazione e cancellazione dell'app e gli aspetti grafici dell'applicazione, come l'icona.
- *Window Manager*, permette di gestire le finestre delle varie applicazioni, controllate da processi diversi, sullo schermo del dispositivo.

### 2.2.3 Native Libraries

Il livello successivo è costituito dalle librerie native realizzate in C e C++ le quali rappresentano l'implementazione di basso livello delle funzionalità offerte dall'*Application Framework*. Tra le principali librerie abbiamo:

- *Media Framework*, componente in grado di gestire i diversi CODEC dei vari formati di acquisizione e riproduzione audio e video. Queste librerie permettono di utilizzare i formati multimediali più popolari, come MPEG4, H.264, MP3, AAC, AMR, JPG, e PNG.
- *Surface Manager*, componente che gestisce le funzionalità del display e coordina le diverse finestre che le applicazioni vogliono visualizzare sullo schermo.
- *Open GL ES*, libreria usata da Android per la grafica che permette di utilizzare sia oggetti 2D che 3D. Può, inoltre, accedere alle funzionalità, se presenti nel dispositivo, di un eventuale acceleratore grafico hardware.
- *SQLite*, libreria che implementa un DBMS relazionale che permette di memorizzare in modo persistente le informazioni e i dati all'interno del dispositivo.
- *WebKit*, browser engine scelto e utilizzato da Android che contiene tutte le funzionalità e le caratteristiche per visualizzare una pagina web all'interno di un'applicazione. È di notevole importanza perché viene adoperato anche da browser come *Safari* e *Chrome*.

## 2.2.4 Android Runtime

L'Android Runtime è localizzato allo stesso livello delle librerie native. È composto dalla *Dalvik Virtual Machine* e dalle librerie che implementano il linguaggio Java.

Android, infatti, utilizza la propria macchina virtuale, non compatibile con la Java Virtual Machine (JVM), chiamata Dalvik, la quale è stata specializzata e ottimizzata per piccoli dispositivi con un processore e una memoria limitata. Per questo motivo, come è possibile vedere dalla Figura 2.6, il codice in Java viene compilato dal compilatore Java in un file `.class` per poi essere convertito in un file `.dex` (*dalvik executable format*) in modo tale da poter essere interpretato dalla Dalvik VM e trasformato in un file eseguibile con estensione `apk`.

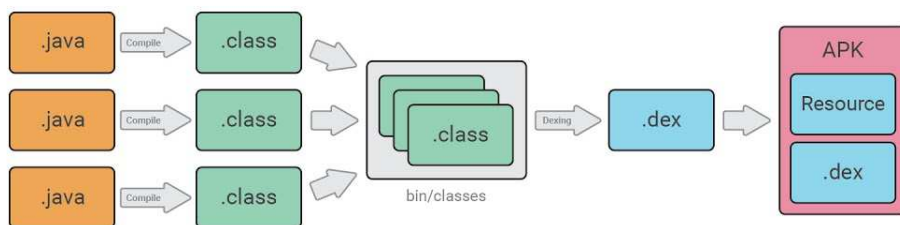


Figura 2.6. I vari passaggi per la compilazione del codice sorgente

## 2.2.5 Linux Kernel

Il livello alla base dell'architettura di Android è il kernel di Linux. Android, pertanto, si appoggia a Linux in quanto si aveva la necessità di costruire un sistema operativo mobile che integrasse funzioni di sicurezza, di gestione della memoria, di gestione dei processi, di power management e che fosse, quindi, molto affidabile e ampiamente testato. Troviamo, inoltre, in questo livello, i driver del dispositivo.

Nella Figura 2.7 viene mostrata l'architettura Android nella sua interezza.

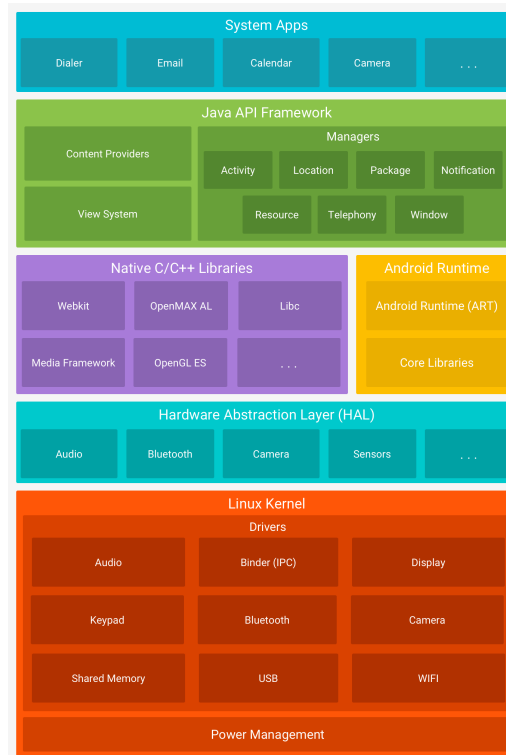
## 2.3 Strumenti per gli sviluppatori

Per realizzare un'applicazione mobile, uno sviluppatore ha bisogno principalmente di tre componenti: *Java Development Kit*, *IDE*, *SDK*.

### 2.3.1 Java Development Kit

Il *Java Development kit*, o JKD, è un insieme di strumenti forniti da Oracle e disponibili gratuitamente sul suo sito ufficiale ([www.oracle.com](http://www.oracle.com)). Esso permette agli sviluppatori di realizzare app in Java. Di conseguenza, visto che il codice sorgente delle app Android è in Java o Kotlin, JDK è un componente principale per il corretto sviluppo di un'app Android.





**Figura 2.7.** Architettura Android

### 2.3.2 Scelta di un IDE: Android Studio

Nel 2008, quando Android 1.0 è stato rilasciato, gli sviluppatori avevano a disposizione solo una serie di strumenti a riga di comando, o script, predefiniti per creare la propria applicazione. La mancanza di funzionalità di un ambiente di sviluppo integrato (IDE) costituiva una barriera per i nuovi sviluppatori, i quali si trovavano in difficoltà con la riga di comando.

Un IDE, infatti, è un ambiente software che permette di facilitare i programmatori fornendo vari componenti, come un editor di codice sorgente che evidenzia la struttura del codice attraverso l'uso di colori, un compilatore, un debugger, il quale permette di analizzare e capire il perché si verificano determinati errori, ed altri strumenti utili.

Per fortuna, alla fine del 2008, furono rilasciati da Google gli strumenti di sviluppo Android (ATD: *Android Development Tools*) per l'IDE *Eclipse* permettendo a questo di diventare l'IDE di riferimento per i progetti Android fino alla presentazione di *Android Studio* nel 2013.

Android Studio, direttamente sviluppato da Google e basato sulla versione gratuita di *IntelliJ IDEA* di JetBrains, divenne, in breve tempo, il punto di riferimento per gli sviluppatori Android. L'IDE viene tuttora arricchito di funzionalità e, con l'ultimo major update che portò la Versione del software dalla 2 alla 3, si è visto un notevole miglioramento. Oltre, infatti, ad una ventata di novità grafiche, vediamo

l'introduzione del linguaggio di programmazione Kotlin, che semplifica di molto la programmazione mobile, ma risulta ancora poco utilizzato a causa della documentazione scarna e per il fatto che non risulta ancora completo sotto il punto di vista delle funzionalità.

### 2.3.3 Android SDK

L'SDK, o *Software Development Kit* (in italiano pacchetto di sviluppo del software) è un insieme di strumenti che aiutano il programmatore nello sviluppo. Questo viene direttamente scaricato durante l'installazione di Android Studio e comprende un insieme di librerie, un emulatore portatile, la documentazione delle varie API e codici esemplificativi delle principali funzionalità implementabili.

## 2.4 Le componenti di un'applicazione Android

Un'applicazione Android è costituita da cinque elementi principali, i quali svolgono un ruolo fondamentale nel comportamento generale dell'applicazione.

### 2.4.1 Le activity e il loro ciclo di vita

Le activity sono le componenti principali della programmazione Android e rappresentano la schermata di un'app con la quale l'utente può interagire. Sono, infatti, equivalenti alle finestre visualizzate a tutto schermo nel mondo desktop.

Si andrà, quindi, ad utilizzare questo componente facendo ereditare ad esso la classe `android.app.Activity` e inserendo al suo interno vari elementi che compongono un'interfaccia, come pulsanti, campi di testo, immagini, etc. Un'applicazione, in generale, avrà più di un'activity e queste saranno collegate da un particolare componente chiamato *intent*, il quale andrà a rappresentare “la volontà” della determinata activity di compiere un particolare task, come aprirne un'altra activity, effettuare un chiamata, aprire le mappe, etc.

Le activity hanno un proprio *ciclo di vita*, ossia una serie di stati attraverso i quali la loro esistenza passa, come si può vedere dalla Figura 2.8. Il primo metodo che incontriamo è `onCreate`, il quale viene invocato quando l'activity viene avviata la prima volta, mentre `onStart` e `onResume` vengono eseguiti, rispettivamente, quando l'activity si prepara per essere visualizzata e quando si può effettivamente interagire con essa.

Ora che l'activity è in esecuzione (Activity Running), possono susseguirsi tre stati. Il primo prende il nome di `onPause` e si verifica quando l'activity non è più in primo piano, ad esempio viene visualizzato un pop-up. Di conseguenza, si può ritornare ad `onResume`, se il pop-up viene cancellato, e, quindi, l'activity torna in primo piano, oppure possiamo finire in `onStop`, se l'activity non è più visibile ma ancora esiste. Infine, per fare in modo che l'activity venga completamente distrutta dal sistema operativo, si deve invocare il metodo `onDestroy`.

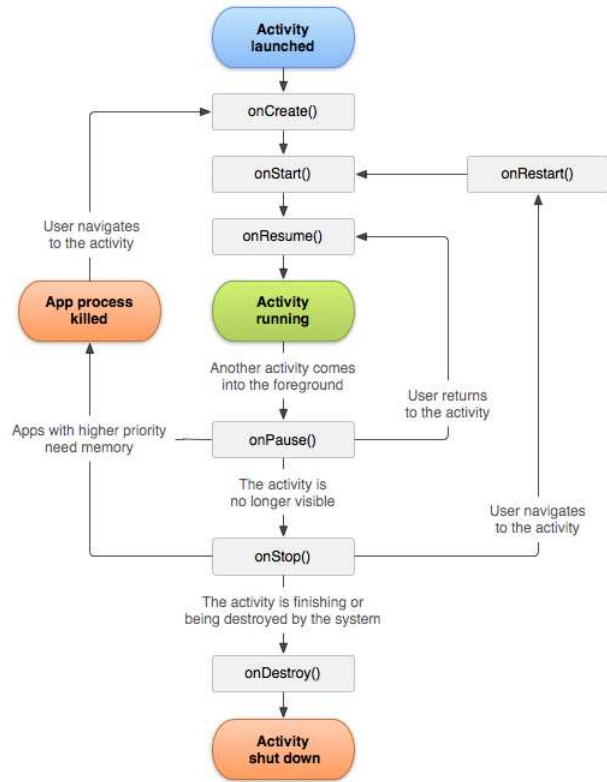


Figura 2.8. Ciclo di vita di un'Activity

### 2.4.2 I Service

I *Service* o, comunemente, in italiano, servizi, rappresentano l'esecuzione di un'attività autonoma in *background* e sono prive di interfaccia grafica. Un esempio di questo tipo di componente consiste nella riproduzione audio, la quale può avvenire in sottofondo mentre parallelamente vengono utilizzate altre applicazioni durante l'ascolto. I Service vengono implementati dalla classe `android.app.Service`.

### 2.4.3 I BroadcastReceiver

I *BroadcastReceiver*, implementabili tramite la classe `android.content.BroadcastReceiver`, permettono di eseguire del codice in risposta ad un determinato evento improvviso come una chiamata in arrivo, la batteria che si è scaricata, connessione dati assente, etc. Questo componente, quindi, permette di implementare una logica nel programma di risposta ad eventi, i quali possono essere generati o da altre applicazioni o dal sistema operativo stesso.

### 2.4.4 I ContentProvider

I *ContentProvider*, implementabili con la classe `android.content`, permettono la condivisione dei dati tra le varie applicazioni. Android, infatti, per garantire la sicurezza e l'integrità del sistema operativo, permette alle applicazioni di essere eseguite solo sul proprio processo, ovvero sulla propria macchina virtuale. Per questo motivo ogni applicazione può essere considerata come un oggetto a se stante e non comunica con le altre applicazioni in automatico.

### 2.4.5 Il Manifest

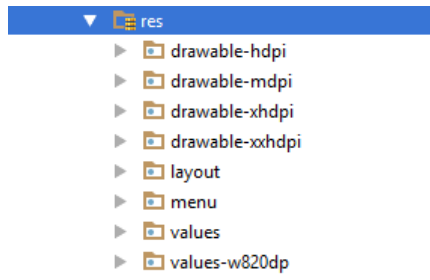
Ogni applicazione ha un file chiamato `AndroidManifest.xml` di tipo XML (*eXtensible Markup Language*) il quale definisce la struttura principale dell'applicazione. Infatti, ogni componente dell'applicazione deve essere registrata in questo file affinché riesca a funzionare; se infatti, l'applicazione non è registrata, il sistema non saprà della sua esistenza e non la eseguirà.

In particolare, all'interno del Manifest troviamo informazioni su:

- *Permessi*: si deve definire quali sono i permessi di cui l'applicazione ha bisogno per funzionare correttamente come, per esempio, se ha bisogno dell'accesso alla memoria interna o esterna, se ha bisogno di accedere ai contatti, etc.
- *Package*: troviamo il nome del pacchetto dell'applicazione il quale rappresenta l'identificatore univoco dell'app.
- *Componenti dell'applicazione*: troviamo elencati tutti gli elementi che compongono l'app, come *Activity*, *Service*, *BroadcastReceiver*, etc. In questo modo Android sa quali componenti possono essere usati e in quale circostanza devono essere eseguiti.
- *API minima*: si deve definire qual è il numero di API minima in cui l'applicazione può essere eseguita.
- *Librerie esterne*: tutte le librerie esterne eccetto quelle contenute nell'*Android Framework API* devono essere dichiarate. Per esempio, *Google Maps Library* deve essere dichiarata nel file Manifest se usata all'interno dell'app.
- *Funzionalità dell'applicazione*: si devono dichiarare anche le funzionalità hardware e software di cui l'applicazione ha bisogno, come il Bluetooth, la fotocamera, etc.

### 2.4.6 Le Risorse

Un'applicazione, infine, potrebbe aver bisogno di elementi multimediali, come immagini, audio o video, e questi possono essere inseriti all'interno dell'applicazione in una apposita cartella chiamata `res`. Questa cartella (Figura 2.9) contiene, a sua volta, una serie di sottocartelle per ogni tipo di risorsa. Per esempio, un file immagine sarà posizionato in `/res/drawable`, una stringa in `/res/value`, mentre un file di layout XML nella directory `/res/layout`.



**Figura 2.9.** Cartella `res` all'interno di un file di progetto



## Analisi dei requisiti

*In questo capitolo effettueremo una descrizione generale del progetto, elencando i vari requisiti funzionali e non funzionali, fino ad arrivare a descrivere gli attori ed i vari casi d'uso dell'applicazione.*

### 3.1 Descrizione del progetto

L'applicazione da sviluppare permetterà al cliente, seduto al ristorante, di effettuare un'ordinazione in completa autonomia.

Il personale del locale, infatti, non dovrà più occuparsi di prendere l'ordinazione, la quale, ora, sarà effettuata direttamente dal cliente, ma dovrà solo fornire a quest'ultimo l'applicazione configurata con il rispettivo numero del tavolo.

Di conseguenza, il compito principale dell'applicazione sarà quello di accompagnare l'utente durante l'ordine, aiutandolo a scegliere il piatto che preferisce, fornendo sia un'immagine esaustiva e veritiera della pietanza, sia informazioni generali, come la descrizione e gli ingredienti del piatto.

L'utente, inoltre, avrà la possibilità di comunicare alla cucina particolari intolleranze o variazioni al piatto, tramite una nota apposita che potrà inserire durante la selezione della pietanza, oppure di personalizzare il piatto, tramite l'inserimento di aggiunte, come, per esempio, mozzarella, prosciutto, alici, etc.

In questo modo, si permetterà al cliente di scegliere cosa ordinare con calma e, inoltre, si cercherà di evitare incomprensioni e/o errori sulla natura del piatto, anche grazie all'utilizzo delle immagini per descriverlo e alla possibilità di visionare tutto l'ordine nel carrello prima di inviarlo alla cucina.

Quest'app, inoltre, permetterà una fidelizzazione del cliente, in quanto esso potrà registrarsi e, di conseguenza, effettuare il login ogni volta che la utilizzerà. L'utente registrato avrà la possibilità di visualizzare il numero di ordini effettuati, gli ordini ancora da pagare e, in particolare, il numero di punti che ha totalizzato nel suo account. Infatti, per ogni 10 euro di spesa, l'utente acquisirà un punto, del valore di 1 euro, che potrà utilizzare sul prossimo acquisto.

Si avrà la necessità, inoltre, di creare un'app con un'interfaccia pulita e semplice in modo tale da essere facilmente usabile da tutte le diverse tipologie di clienti. Si

è deciso, comunque, di fornire la possibilità al cliente di chiedere aiuto tramite un particolare pulsante all'interno dell'app.

In conclusione, l'utilità dell'applicazione sarà quella di ridurre il tempo di attesa del cliente durante il servizio riuscendo a facilitare ed alleggerire il compito dei camerieri; allo stesso tempo, verrà fornito al cliente un metodo per ordinare immediatamente, senza dover aspettare l'arrivo di un cameriere.

## 3.2 Requisiti funzionali e non funzionali

Un'attività fondamentale durante la stesura dei requisiti è quella di elencare e individuare i vari requisiti funzionali e non funzionali che l'applicazione dovrà garantire. Ciò permette una visione d'insieme delle funzionalità che l'applicazione deve rispettare e consente agli sviluppatori di comprendere il dominio applicativo nella sua interezza, cercando di evitare incomprensioni che potrebbero portare a cambiamenti della logica dell'app durante lo sviluppo.

### 3.2.1 Requisiti funzionali

I requisiti funzionali rappresentano l'insieme delle caratteristiche e dei servizi che l'app dovrà fornire. Essi sono indipendenti dalla tecnologia, dalla piattaforma e dal linguaggio di programmazione. Le funzionalità principali che vorremo garantire sono le seguenti:

- *Attività CRUD per i tavoli*: si deve avere la possibilità di modificare il numero del tavolo associato al dispositivo.
- *Attività CRUD per l'ordine*: si deve avere la possibilità di creare e visionare l'ordine effettuato.
- *Attività CRUD per le pietanze*: si deve avere la possibilità di creare o cancellare i vari piatti prima di inviarli alla cucina.
- *Attività CRUD per la sessione*: si deve avere la possibilità di chiudere la sessione corrente ed aprirne una nuova.
- *Attività CRUD per l'utente*: si deve avere la possibilità di registrare e visualizzare le informazioni di un utente.
- *Attività CRUD per i punti cliente*: si deve avere la possibilità di visualizzare, inserire, modificare e cancellare i punti che l'utente ha a disposizione.
- *Richiesta aiuto e Invio messaggi al cameriere*: si deve avere la possibilità di richiedere aiuto per l'ordinazione ed inviare messaggi per notificare la volontà di procedere al pagamento.

### 3.2.2 Requisiti non funzionali

I requisiti non funzionali definiscono le proprietà e i vincoli realizzativi, di tipo tecnico, che il sistema dovrà necessariamente rispettare. I principali requisiti non funzionali sono:



- *Manutenibilità*: si deve avere la possibilità di aggiungere e/o eliminare funzionalità all'applicazione senza compromettere il funzionamento di quelle già esistenti.
- *Usabilità*: si deve implementare l'applicazione in modo da essere facile da comprendere e semplice da utilizzare per le varie categorie di utenza. L'applicazione, infatti, dovrà essere utilizzata sia dai nativi digitali sia da utenti non esperti.
- *Durata sessione*: si deve implementare il concetto di sessione, la quale inizia dopo aver inserito il numero del tavolo e si conclude quando l'utente avrà completato il pagamento degli ordini effettuati.
- *Database online*: si deve utilizzare un database online in modo tale da permettere di visualizzare i dati dei clienti, gli ordini effettuati e i messaggi ricevuti su più dispositivi. Inoltre, si deve avere la possibilità di modificare il menù senza il bisogno di aggiornare l'app.

## 3.3 Attori e casi d'uso

### 3.3.1 Diagramma dei casi d'uso

Il primo tipo di diagramma che viene sviluppato durante l'analisi dei requisiti è quello degli attori e dei casi d'uso. Esso rappresenta coloro che interagiscono con il sistema (attori) e le azioni che essi possono effettuare (casi d'uso).

Un attore, infatti, specifica un ruolo assunto da un utente o da un'altra entità che interagisce con il sistema e viene rappresentato nel diagramma con la forma dell'omino stilizzato. Gli attori presenti nell'app e mostrati in Figura 3.1 sono i seguenti:

- *Utente non registrato*: colui che utilizzerà l'applicazione per ordinare senza effettuare il login.
- *Utente registrato*: colui che, prima di utilizzare l'applicazione per ordinare, avrà effettuato il login e potrà accedere a sconti personalizzati.
- *Cameriere/Staff*: colui che potrà accedere ad un'area riservata la quale gli permetterà di cambiare il numero del tavolo e chiudere la sessione corrente indicando l'effettivo pagamento dell'utente.

Un caso d'uso, invece, specifica un insieme di azioni che producono un risultato osservabile per uno o più attori. I casi d'uso vengono rappresentati nel diagramma per mezzo di ovali e la descrizione all'interno dovrebbe essere basata su un verbo o su un sostantivo esprimente un avvenimento. I diagrammi dei casi d'uso relativi all'applicazione sviluppata sono mostrati nelle Figure 3.2-3.4.

### 3.3.2 Tabella dei casi d'uso

I casi d'uso utilizzati nell'applicazione sono elencati di seguito:

- inserimento e visualizzazione di un ordine;
- richiesta di un cameriere;
- richiesta del pagamento;

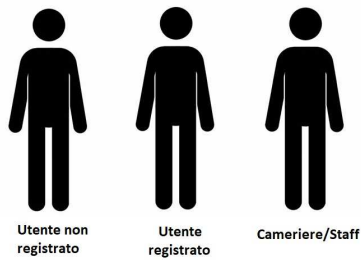


Figura 3.1. Elenco degli attori

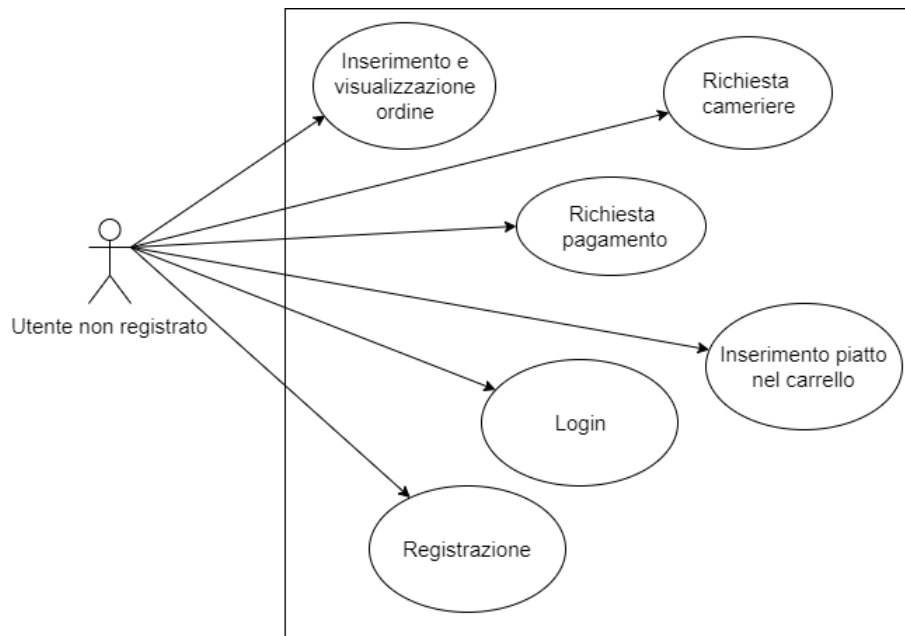
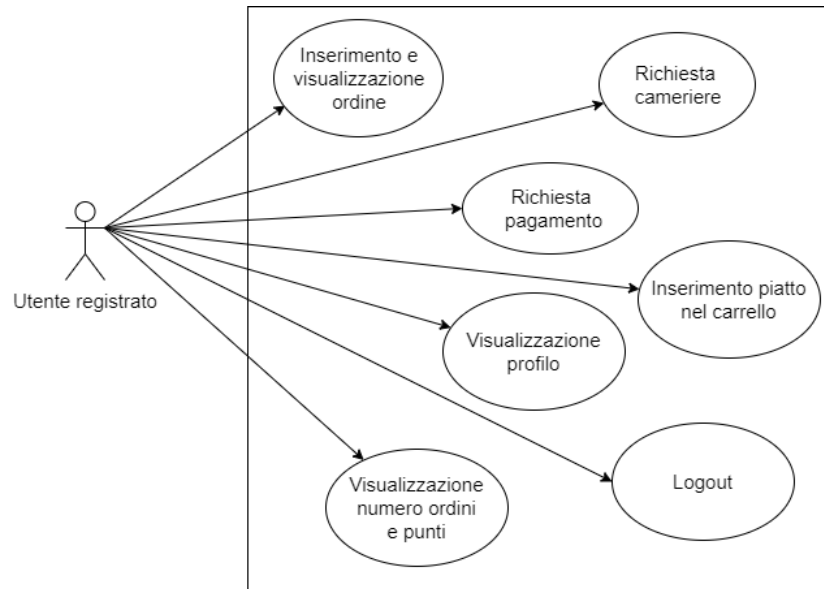
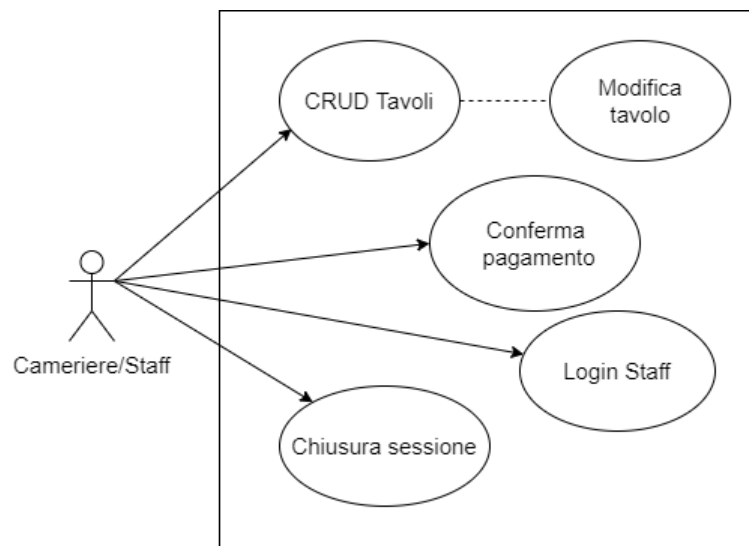


Figura 3.2. Diagramma dei casi d'uso relativo all'utente non registrato

- inserimento del piatto nel carrello
- login;
- registrazione;
- visualizzazione del profilo;
- logout;
- visualizzazione del numero di ordini e dei punti;
- CRUD dei tavoli;
- modifica di un tavolo;
- conferma di un pagamento;
- login da parte dello Staff;
- chiusura di una sessione.



**Figura 3.3.** Diagramma dei casi d'uso relativo all'utente registrato



**Figura 3.4.** Diagramma dei casi d'uso relativo al Cameriere/Staff del ristorante

### 3.4 Matrice di mapping

La *matrice di mapping*, o di tracciabilità, è un componente fondamentale che permette di mantenere sotto controllo l'evoluzione dei requisiti, in quanto mette in relazione un requisito funzionale con più casi d'uso. Essa può, inoltre, essere usata per verificare se i requisiti di progetto correnti sono stati effettivamente raggiunti.

La matrice, infatti, permette di evidenziare se un requisito sia coperto da più casi d'uso oppure, se ad alcuni requisiti non è associato nessun caso d'uso.

In Figura 3.5 viene mostrata la matrice di mapping dell'applicazione.

REQUISITI FUNZIONALI		CASI D'USO											
		Inserimento e visualizzazione ordine	Richiesta cameriere	Richiesta pagamento	Inserimento piatto nel carrello	Login	Registrazione	Visualizzazione profilo	Logout	Visualizzazione numero ordini e punti	CRUD Tavoli	Conferma pagamento	Chiusura sessione
CRUD Tavoli											X		
CRUD Ordini	X									X		X	
CRUD Pietanze			X										
CRUD Sessione													X
CRUD Utente						X	X	X	X				
CRUD Punti Cliente										X			
Richiesta aiuto ed invio messaggio al cameriere		X	X										

Figura 3.5. Matrice di mapping relativa all'applicazione

## Progettazione

*In questo capitolo affronteremo i principali passaggi necessari per la progettazione di un'applicazione mobile. Nella prima parte, attraverso una mappa dell'applicazione ed i vari mockup, rappresenteremo la struttura dell'app, mentre, nella seconda parte, illustreremo in dettaglio le componenti necessarie per la progettazione della base di dati.*

### 4.1 Struttura dell'applicazione

#### 4.1.1 Mappa dell'applicazione

La mappa dell'applicazione, mostrata in Figura 4.1, permette di schematizzare il funzionamento dell'applicativo in un unico diagramma. I vari livelli dell'applicazione, infatti, vengono rappresentati tramite blocchi rettangolari e vengono divisi secondo vari livelli funzionali per mezzo dei colori. I blocchi sono collegati tra di loro con frecce orientate, le quali permettono di indicare il corretto ordine di lettura della sequenza.

Con il colore **giallo** indichiamo la home page della nostra applicazione, la quale è rappresentata dal blocco *Home* nel normale utilizzo, oppure da *Imposta Numero Tavolo*, se non abbiamo collegato all'app un tavolo corrispondente.

L'utente, una volta aperta l'app, potrà muoversi facilmente, per mezzo di una navbar, tra le tre principali macroaree, chiamate *Home*, *Ordina*, *Profilo*, rappresentate con il colore **verde**. Queste gli permetteranno di raggiungere tutte le principali attività dell'applicazione, evidenziate in **arancione**.

Si è deciso, inoltre, di evidenziare con il colore **viola** i passaggi intermedi della singola attività, mentre con il colore **blu** il blocco che permette di concludere l'attività. Infine, i blocchi di colore **grigio** permettono una migliore comprensione delle funzioni svolte durante il passaggio al blocco successivo.

In conclusione, come è possibile notare dalla mappa dell'app, i percorsi ideati, che saranno effettuati dall'utente, sono semplici e lineari. In questo modo, si riesce ad ottenere un'applicazione semplice dal punto di vista funzionale e si ha la sicurezza che le funzionalità inserite saranno implementate nella macroarea più pertinente.

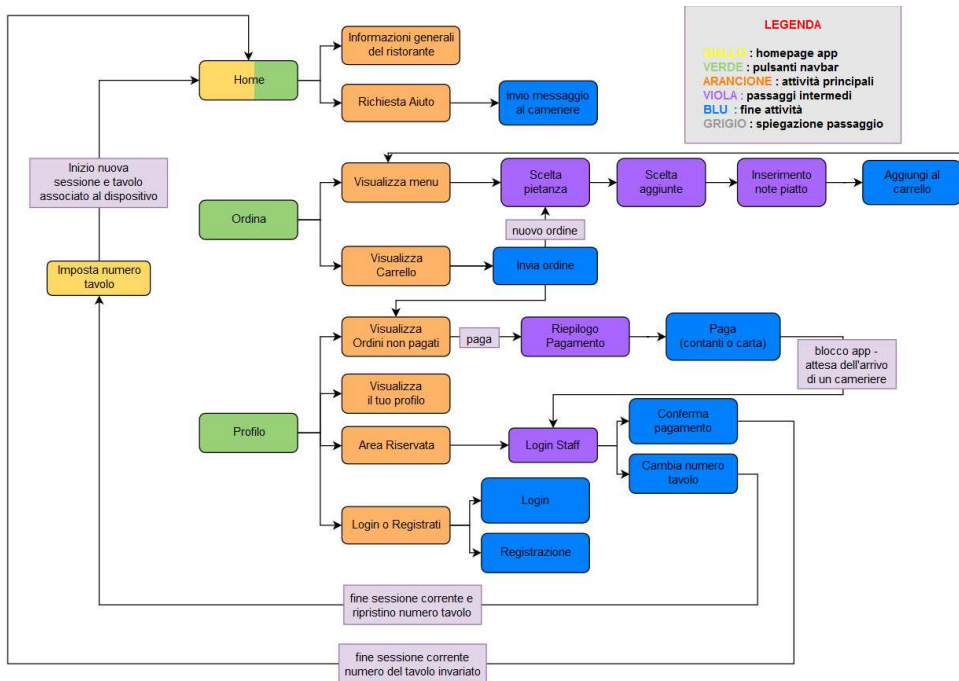


Figura 4.1. Mappa relativa all'applicazione mobile

### 4.1.2 Mockup

La realizzazione dei *mockup* è una fase importante e necessaria da svolgere durante la progettazione di un'applicazione. Essa permette, infatti, in poco tempo, di delineare la struttura e gli elementi grafici che dovranno essere presenti all'interno delle singole pagine dell'app.

Il principale scopo per cui si utilizzano i mockup è perché essi permettono, in un colpo d'occhio e senza scrivere codice, di visualizzare tutta l'interfaccia grafica dell'app e, di conseguenza, di creare alternative diverse della stessa pagina per capire quale sia quella più apprezzata.

Nelle Figure 4.2-4.5 vengono riportate le quattro principali schermate dell'applicazione sviluppata. La prima (Figura 4.2) permette di associare al dispositivo il numero del tavolo corrente, mentre le restanti sono quelle relative alle pagine *Home* (Figura 4.3), *Ordina* (Figura 4.4) e *Profilo* (Figura 4.5), accessibili per mezzo della navbar.

Inoltre, nelle Figure 4.6-4.8 evidenziamo il percorso che l'utente dovrà effettuare per aggiungere un piatto al carrello. Egli, infatti, dopo aver scelto il piatto nella scheda *Pietanza* (Figura 4.6), potrà aggiungere ad esso ulteriori ingredienti tramite la scheda *Aggiunta* (Figura 4.7), oppure, tramite la scheda *Note* (Figura 4.8), potrà inserire una nota sul piatto corrispondente.

Inoltre, la Figura 4.9 rappresenta l'interfaccia grafica della scheda *Carrello*, mentre le Figure 4.10-4.12 illustrano il percorso che l'utente deve effettuare per pagare le ordinazioni non saldate.

In ultimo, esponiamo le restanti schede di minore importanza, ovvero

- la scheda *Login Utente* con la quale l'utente può autenticarsi (Figura 4.13);
- la scheda *Registrazione* (Figura 4.14);
- la scheda *Visualizza Informazioni*, dove l'utente può visualizzare le informazioni che l'applicazione ha in suo possesso (Figura 4.15);
- la scheda *Area Riservata*, nella quale il cameriere può confermare il pagamento o cambiare il numero del tavolo (Figura 4.16);
- la scheda *Pagamento* nella variante visualizzata dall'utente registrato; in essa sono mostrati i punti in possesso dell'utente e il nuovo costo totale dell'ordine dopo l'utilizzo dei punti (Figura 4.17);
- la scheda *Profilo* nella variante visualizzata dall'utente registrato (Figura 4.18);



**Figura 4.2.** Mockup relativo alla scheda “Imposta Tavolo”

### 4.1.3 Flow Chart o Diagrammi di Flusso

Per rappresentare in modo chiaro le varie schermate che l'utente incontrerà nel normale utilizzo dell'applicazione, si è ritenuto efficace utilizzare i *Diagrammi di Flusso* o *Flow Chart*. Questi sono degli strumenti molto utili perché, oltre a consentire una migliore comprensione dell'applicativo, permettono di capire se la logica dell'applicazione sviluppata sia coerente e priva di errori. I diagrammi sono costituiti da frecce unidirezionali e da un punto di ingresso e un punto di uscita, che rappresentano l'inizio e la fine dell'attività mostrata. Nelle Figure 4.19-4.21 vengono presentati i Diagrammi di Flusso dell'applicazione sviluppata.

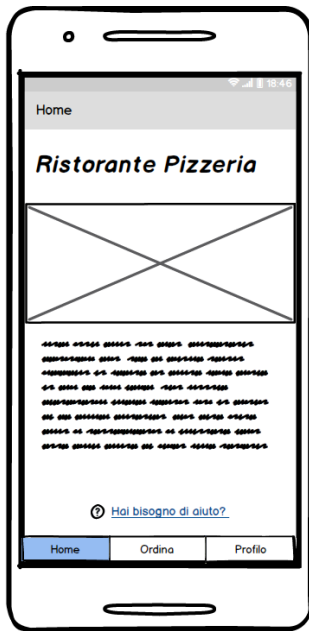


Figura 4.3. Mockup relativo alla scheda "Home"

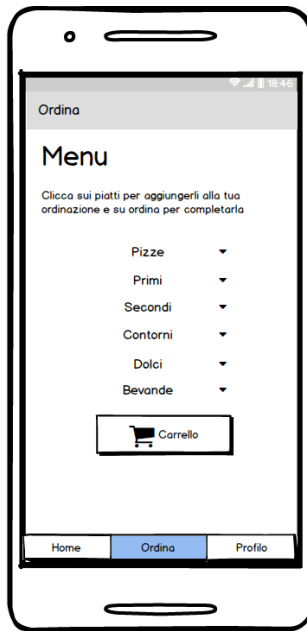


Figura 4.4. Mockup relativo alla scheda "Ordina"



Figura 4.5. Mockup relativo alla scheda "Profilo"

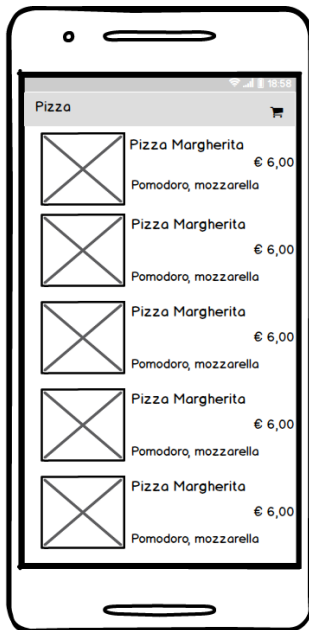


Figura 4.6. Mockup relativo alla scheda "Pietanza"



Figura 4.7. Mockup relativo alla scheda "Aggiunte"



Figura 4.8. Mockup relativo alla scheda "Note"





Figura 4.9. Mockup relativo alla scheda “Carrello”



Figura 4.10. Mockup relativo alla scheda “Comande”

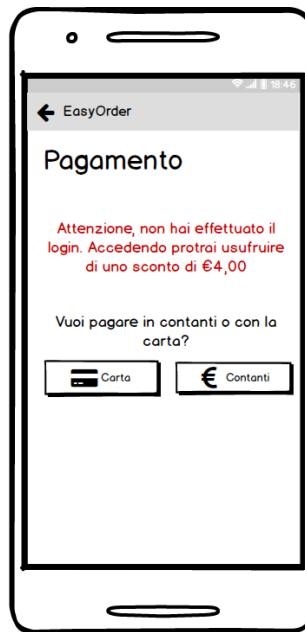


Figura 4.11. Mockup relativo alla scheda “Pagamento”

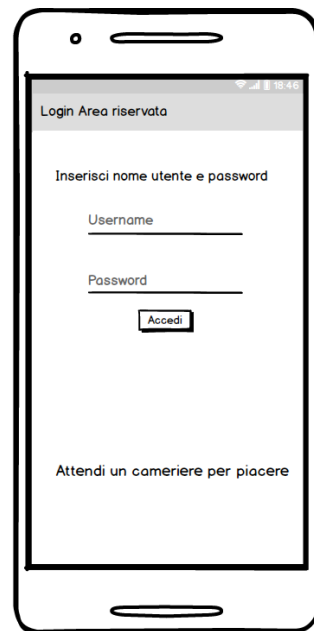


Figura 4.12. Mockup relativo alla scheda “Login Staff”



Figura 4.13. Mockup relativo alla scheda “Login Utente”



Figura 4.14. Mockup relativo alla scheda “Registrazione”



Figura 4.15. Mockup relativo alla scheda “Visualizza Informazioni”

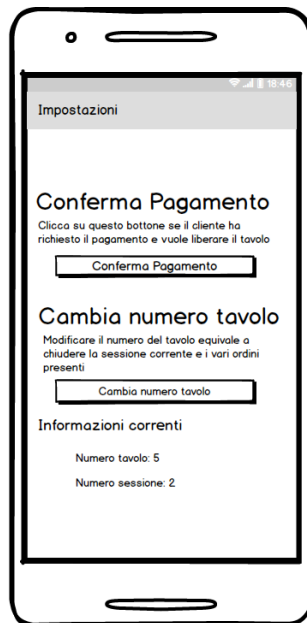


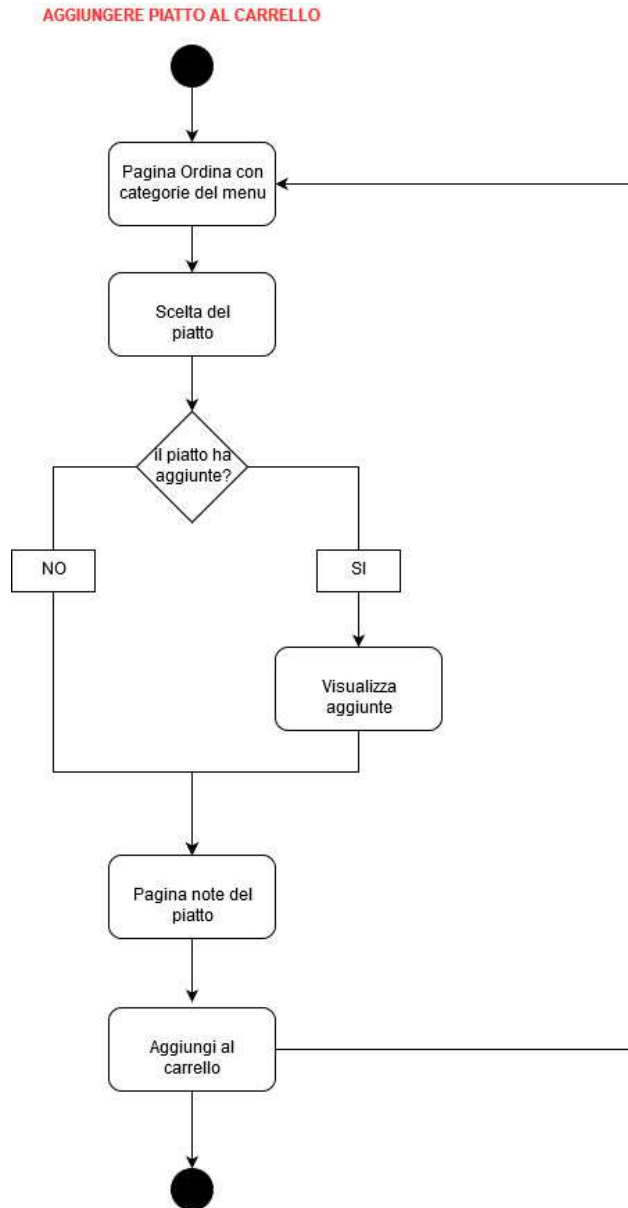
Figura 4.16. Mockup relativo alla scheda “Area Riservata”



Figura 4.17. Mockup relativo alla scheda “Pagamento” per un utente registrato



Figura 4.18. Mockup relativo alla scheda “Profilo” per un utente registrato



**Figura 4.19.** Diagramma di attività relativo alla scelta del piatto da aggiungere al carrello

## 4.2 Progettazione della base di dati

Progettare una base di dati consiste nell'individuare e nel definire i dati che l'applicazione memorizzerà. Esistono, principalmente, due fasi per progettare correttamente una base di dati; esse prendono il nome di *progettazione concettuale* e *progettazione logica*.

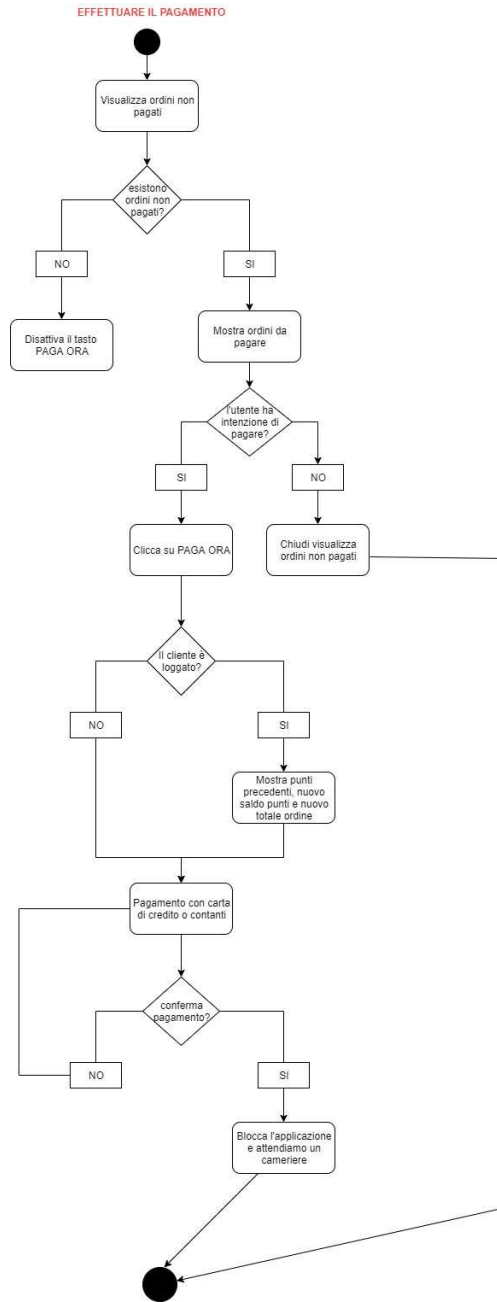


Figura 4.20. Diagramma di attività relativo alla richiesta del pagamento degli ordini

### 4.2.1 Progettazione concettuale

La *progettazione concettuale* ha lo scopo di rappresentare la realtà di interesse del nostro applicativo in modo preciso e conciso ma, comunque, indipendente dai criteri

di rappresentazione usati dal sistema informatico scelto per l'implementazione. Essa permette, quindi, di ottenere una rappresentazione astratta, ma fedele, della realtà.

Il primo passo da effettuare è l'individuazione delle entità principali coinvolte nel nostro applicativo. Grazie agli elementi come *Flow Chart* e alla *Mappa dell'applicazione*, si è ottenuto una visione di insieme e si sono individuate 4 principali macroaree, rappresentate in Figura 4.22. Più specificatamente, le quattro macroaree individuate sono:

- *Utente*: rappresenta il macroblocco che contiene tutte le informazioni fornite dall'utente al momento della registrazione, come il nome, il cognome e l'e-mail.
- *Sessione Tavolo*: rappresenta il macroblocco che contiene tutte le informazioni che permettono di identificare il tavolo e l'istante temporale in cui l'utente ha effettuato l'ordinazione.
- *Ordine*: rappresenta il macroblocco relativo alle comande, ossia all'insieme di piatti che compongono un ordine. Esso è caratterizzato da elementi come il costo totale e il testo dell'ordine.
- *Voce Menù*: rappresenta il macroblocco che corrisponde alla pietanza all'interno del menù. Essa è caratterizzata da elementi come il nome, la foto e il prezzo.

Una volta individuati i macroblocchi che compongono il nostro applicativo, il passo successivo è sviluppare tali macroblocchi ed integrarli fra di loro per ottenere uno schema fondamentale nella progettazione, chiamato *Diagramma E/R*.

### Diagramma E/R

Il *Diagramma E/R*, chiamato, anche, in italiano *Schema Entità/Relazione* fa uso del modello E/R, un modello teorico per la rappresentazione concettuale e grafica dei dati, proposto dal professore Peter Chen nel 1976.

I principali costrutti di cui è composto tale diagramma sono: le entità, le relazioni e gli attributi.

Le *entità* rappresentano classi di oggetti con esistenza autonoma e proprietà comuni e vengono rappresentate nel Diagramma E/R per mezzo di rettangoli. Le *relazioni* o *associazioni*, invece, rappresentano legami logici tra due o più relazioni e vengono rappresentate graficamente con dei rombi. Infine, sia le entità che le relazioni possono essere descritte utilizzando degli *attributi*, i quali rappresentano le proprietà elementari dell'oggetto.

In Figura 4.23 viene mostrato il diagramma E/R dell'applicativo sviluppato.

Come si evince dalla figura, l'applicazione dovrà, prima di tutto, memorizzare le informazioni relative ai piatti che compongono il menù. Queste saranno contenute all'interno dell'entità *Voce Menù*, la quale rappresenta il singolo piatto, e ha come chiave un *ID* e come attributi il *Nome*, il *Prezzo*, gli *Ingredienti*, la *Categoria* a cui appartiene (Pizze, Primi, Secondi, Contorni, Dolci, Bevande), il *Link dell'Immagine* e la *Posizione* in cui vogliamo che il piatto venga visualizzato nella lista. Ogni piatto, inoltre, potrà essere associato, tramite la relazione *Inserite*, ad uno o più elementi dell'entità *Aggiunta* la quale rappresenta i vari ingredienti aggiuntivi con cui l'utente può personalizzarlo. Quest'entità avrà come chiave un *ID*, mentre avrà come attributi il *Nome*, la *Categoria* a cui appartiene e il *Prezzo*. La cardinalità della relazione da parte di *Voce Menù* è (0,N), in quanto un piatto potrebbe non

avere nessuna aggiunta, mentre dalla parte di *Aggiunta* è (1,N), perché un'aggiunta deve essere associata almeno ad un piatto.

In *Utente*, invece, saranno memorizzati i dati che l'utente fornisce durante la registrazione. L'entità sarà formata da una chiave *ID Utente* e dagli attributi *Email*, *Numero Punti*, che rappresenta il totale dei punti che l'utente può utilizzare nel prossimo ordine; infine, abbiamo deciso di introdurre la ridondanza *Numero ordini*, la quale rappresenta il numero totale degli ordini effettuati dal cliente.

L'entità *Utente* è collegata all'entità *Ordine* tramite la relazione *Effettua*. L'utente, infatti, può effettuare degli ordini i quali avranno come chiave un *TimeStamp*, ossia l'orario in cui è stato effettuato l'ordine, e la chiave esterna *Numero sessione* dell'entità *Sessione Tavolo*. Per quanto riguarda, invece, gli attributi si è scelto di memorizzare il *Testo dell'Ordine*, se l'utente ha *Pagato* o no il conto, e, infine, si è introdotta la ridondanza *Costo Totale*, perché, essendo quest'ultimo un dato che il ristorante utilizza spesso, sarebbe stato troppo dispendioso calcolarlo ad ogni utilizzo. La cardinalità della relazione *Effettua* è (0,N), dalla parte di *Utente*, perché un utente potrebbe non effettuare nessun ordine, mentre, dalla parte di *Ordine*, è (1,1), perché un ordine è associato ad un solo utente.

L'entità *Utente* è associata, tramite la relazione *Invia*, all'entità *Notifica*, la quale rappresenta i messaggi che l'utilizzatore dell'app può inviare ai camerieri. Tale entità ha come attributo il *Corpo del Messaggio* e come chiave un *Timestamp*, unita alla chiave esterna *Numero Sessione* dell'entità *Sessione Tavolo*. Per quanto riguarda la cardinalità della relazione *Invia* troviamo che questa è (0,N) dalla parte di *Utente*, perché un utente potrebbe non inviare nessun messaggio, mentre, dalla parte di *Notifica*, è pari a (1,1), perché un messaggio è associato ad un solo utente.

Nell'entità *Sessione Tavolo*, inoltre, sono memorizzate le sessioni create dagli utenti dell'app; tale entità ha come chiave il *Numero Sessione*, mentre come attributi il *Numero del Tavolo*, l'*Ora di inizio* e l'*Ora di fine* della sessione. Quest'ultima entità, a sua volta, è collegata all'entità *Notifica*, tramite la relazione *Associa*, e all'entità *Ordine*, tramite la relazione *Ordina*. La prima ha cardinalità (1,1) dalla parte di *Notifica*, in quanto un messaggio è associato ad una sola sessione mentre ha cardinalità pari a (0,N) dalla parte di *Sessione Tavolo* perché l'utente di una sessione può inviare zero o più messaggi. La seconda ha cardinalità (0,N) dalla parte di *Sessione Tavolo*, perché ad una sessione potrebbe non essere associato nessun ordine, e (1,1) dalla parte di *Ordine*, perché ad un ordine deve essere associata una ed una sola sessione.

L'entità *Staff*, infine, rappresenta il personale del ristorante, il quale ha come chiave l'*ID* e come attributi lo *User*, ossia il nome utente, e la *Password*.

### Dizionario delle entità e dizionario delle relazioni

Il *dizionario dei dati* è un ulteriore strumento utile durante la progettazione di una base di dati perché permette di arricchire lo schema E/R con descrizioni in linguaggio naturale. In particolare, nelle Figure 4.24 e 4.25 vengono mostrate le due tabelle relative ai dizionari delle entità e delle relazioni. Le tabelle, oltre a descrivere le entità o le relazioni, permettono di elencare gli attributi associati ad esse e i loro identificatori.

### Vincoli di integrità

I *vincoli di integrità* sono dei vincoli che non possono essere rappresentati nello schema E/R perché hanno bisogno di una descrizione in linguaggio naturale. Identificarli permette di evitare situazioni che potrebbero portare in errore l'applicativo; per questo motivo, vengono sempre implementati nella base di dati, in quanto evitano in anticipo la creazione di istanze errate. Tali vincoli sono, quindi, uno strumento utile per garantire l'integrità del DBMS.

Di seguito elenchiamo i vari vincoli di integrità della base di dati del nostro applicativo:

- *V1*: il campo “Prezzo”, relativo all’entità “Voce Menù”, deve essere un numero maggiore di 0.
- *V2*: il campo “Prezzo”, relativo all’entità “Aggiunta”, deve essere un numero maggiore di 0.
- *V3*: il campo “Costo Totale”, relativo all’entità “Ordine”, deve essere maggiore di 0.
- *V4*: l’attributo “Pagato”, relativo all’entità “Ordine”, deve essere 0 (non pagato) o 1 (pagato).
- *V5*: l’attributo “Posizione”, relativo all’entità “Voce Menù”, deve essere un numero maggiore di 0.
- *V6*: l’attributo “Categoria”, relativo all’entità “Voce Menù” e all’entità “Aggiunta”, può assumere soltanto i seguenti valori: “Pizze”, “Primi”, “Secondi”, “Contorni”, “Dolci” e “Bevande”.
- *V7*: l’attributo “Ingredienti”, relativo all’entità “Voce Menù”, è opzionale.
- *V8*: l’attributo “Numero Sessione”, relativo all’entità “Sessione Tavolo”, deve essere un numero maggiore di 0.
- *V9*: l’attributo “Numero Tavolo”, relativo all’entità “Sessione Tavolo”, deve essere un numero maggiore di 0.
- *V10*: l’attributo “Ora fine”, relativo all’entità “Sessione Tavolo”, deve essere maggiore dell’attributo “Ora inizio”.
- *V11*: l’attributo “E-mail”, relativo all’entità “Utente”, deve essere univoco.
- *V12*: l’attributo “Numero ordini”, relativo all’entità “Utente”, deve essere maggiore o uguale a 0.
- *V13*: l’attributo “Numero punti”, relativo all’entità “Utente”, deve essere maggiore o uguale a 0.

### Elenco identificatori principali

In Figura 4.26 vengono mostrati gli identificatori principali delle entità dello schema E/R. Gli identificatori rappresentano le chiavi univoche della base di dati. Per ciò che riguarda il nostro applicativo, si è ritenuto opportuno utilizzare degli ID per l’entità *Aggiunta*, *Staff* e *Voce Menù*, mentre, per *Notifica* e *Ordine*, l’identificativo consiste nel numero della sessione e nella data prelevata dall’orologio nel momento dell’ordine o dell’invio del messaggio.



### Glossario dei termini

In Figura 4.27 viene mostrato il glossario dei termini. Questo viene utilizzato principalmente per spiegare in linguaggio naturale alcuni termini che potrebbero essere fraintesi durante l'interpretazione dello schema E/R. Nella tabella, infatti, forniamo, per ogni termine, una breve descrizione e i possibili sinonimi utilizzati durante la descrizione in linguaggio naturale dell'applicativo.

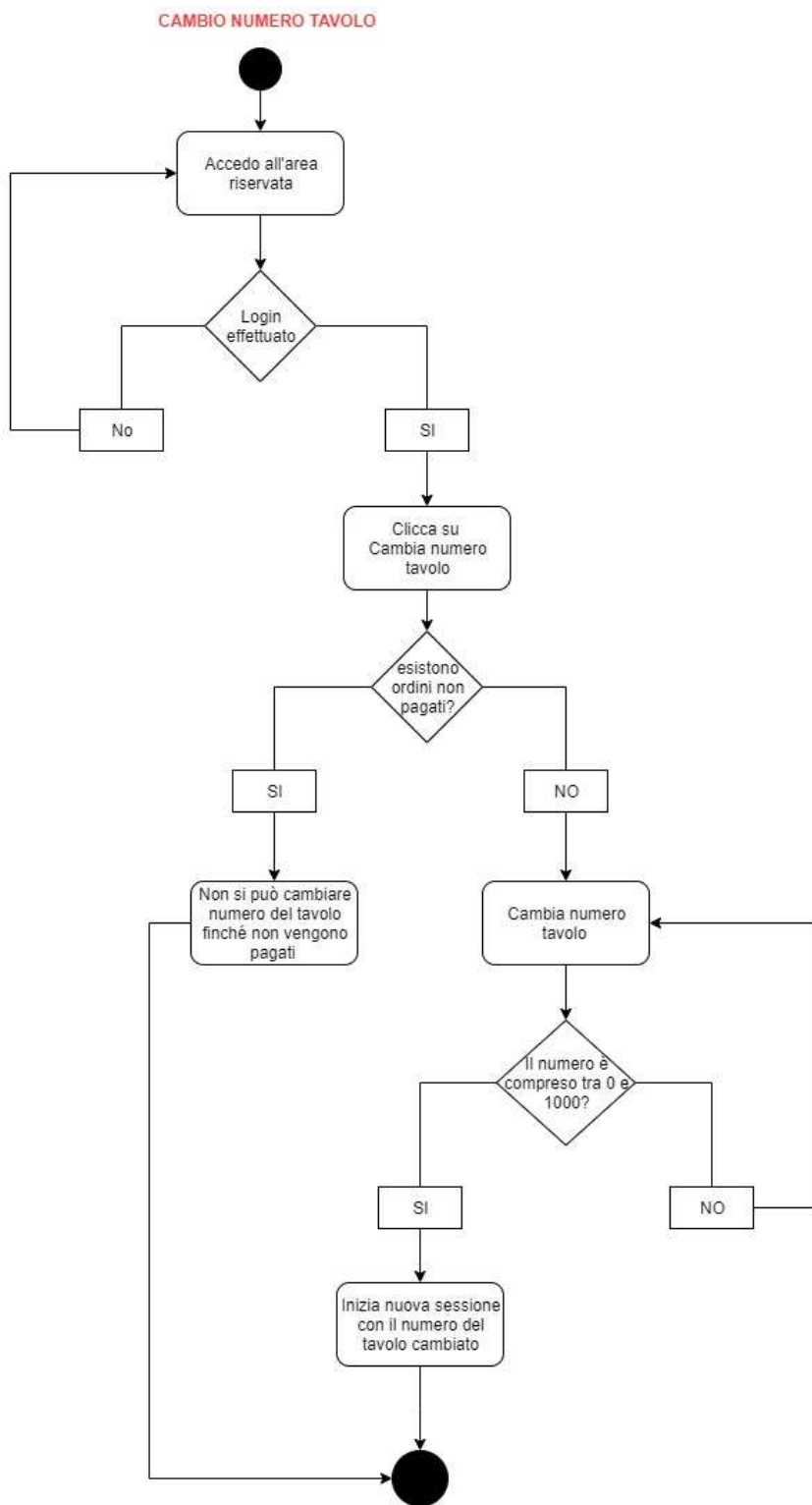
## 4.2.2 Progettazione logica

### Traduzione verso il modello relazionale

Dopo aver definito il diagramma E-R frutto della progettazione concettuale, si riesce, in modo semplice e immediato, a definire uno schema logico, che tiene conto del tipo di database che si vuole utilizzare. Nel nostro caso si è deciso di adottare un *database relazionale*. Sarà, quindi, necessario trasformare lo schema concettuale in uno schema relazionale equivalente, il quale sarà in grado di rappresentare le medesime informazioni utilizzando, però, delle opportune tabelle.

L'idea alla base della traduzione è che le entità diventano tabelle con gli stessi attributi, mentre le relazioni diventano tabelle con gli identificatori delle entità coinvolte.

Nella Figura 4.28 viene mostrata la traduzione verso il modello relazionale dell'applicativo da noi sviluppato.



**Figura 4.21.** Diagramma di attività relativo alla funzionalità del cambio numero del tavolo



Figura 4.22. Entità principali dell'applicativo sviluppato

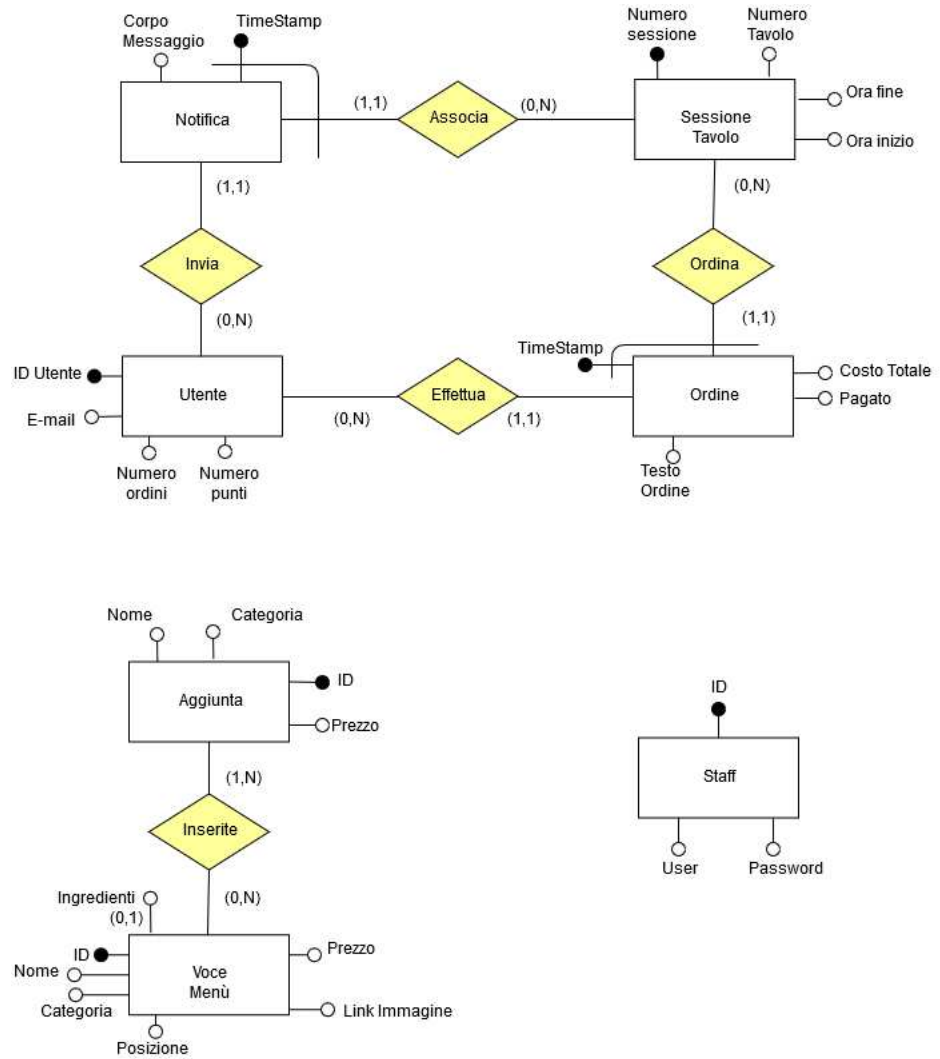


Figura 4.23. Diagramma E/R relativo all'applicativo sviluppato

<b>NOME ENTITÀ</b>	<b>DESCRIZIONE</b>	<b>ATTRIBUTI</b>	<b>IDENTIFICATORE</b>
<i>Aggiunta</i>	Rappresenta l'insieme degli ingredienti che possono essere aggiunti ai piatti del menù	ID (numerico), Nome (stringa), Categoria (stringa), Prezzo (numerico)	ID (numerico)
<i>Notifica</i>	Raccolta dei messaggi inviati dall'utente al personale	TimeStamp (data), Corpo Messaggio (stringa)	TimeStamp (data), Numero Sessione di "Sessione Tavolo"
<i>Ordine</i>	Rappresenta l'ordine effettuato dal cliente	TimeStamp (data), Costo Totale (numerico), Pagato (0,1), Testo Ordine (stringa)	TimeStamp (data), Numero Sessione di "Sessione Tavolo"
<i>Staff</i>	Rappresenta il personale del locale	ID (numerico), User (stringa), Password (stringa)	ID (numerico)
<i>Sessione Tavolo</i>	Rappresenta la sessione effettiva del cliente collegata al numero del tavolo a cui fa riferimento	Numero Sessione (numerico), Numero Tavolo (numerico), Ora inizio (data), Ora fine (data)	Numero Sessione (numerico)
<i>Utente</i>	Rappresenta l'identità dell'utente che ha effettuato la registrazione	ID Utente (stringa), E-mail (stringa), Numero Ordini (numerico), Numero Punti (numerico)	ID Utente (stringa)
<i>Voce Menù</i>	Rappresenta i piatti che il cliente può ordinare	ID (numerico), Nome (stringa), Ingredienti (stringa), Categoria (stringa), Posizione (numerico), Prezzo (numerico), Link Immagine (stringa)	ID (numerico)

Figura 4.24. Dizionario delle entità relativo alla base di dati dell'applicazione

<b>NOME RELATIONSHIP</b>	<b>DESCRIZIONE</b>	<b>ENTITÀ COINVOLTE</b>	<b>ATTRIBUTI</b>
<i>Associa</i>	Associa la notifica con la sessione che le ha inviate	Notifica (1,1), Sessione Tavolo (0,N)	\\
<i>Effettua</i>	Associa l'ordine all'utente che lo ha richiesto	Ordine (1,1), Utente (0,N)	\\
<i>Inserite</i>	Associa le aggiunte possibili ai piatti del menù	Aggiunta (1,N), Voce Menù (0,N)	\\
<i>Invia</i>	Associa la notifica all'utente che le ha inviate	Notifica (1,1), Utente (0,N)	\\
<i>Ordina</i>	Associa l'ordine alla sessione che lo ha effettuato	Ordine (1,1), Sessione Tavolo (0,N)	\\

Figura 4.25. Dizionario delle relazioni relativo alla base di dati dell'applicazione

<b>NOME ENTITÀ</b>	<b>IDENTIFICATORE</b>
<i>Aggiunta</i>	ID
<i>Notifica</i>	TimeStamp, Numero Sessione
<i>Ordine</i>	TimeStamp, ID Utente
<i>Sessione Tavolo</i>	Numero Sessione
<i>Staff</i>	ID

**Figura 4.26.** Elenco degli identificatori principali relativo alla base di dati dell'applicazione

<b>TERMINE</b>	<b>DESCRIZIONE</b>	<b>SINONIMO</b>
<i>Categoria</i>	Rappresenta la categoria del piatto e può essere "Pizze", "Primi", "Secondi", "Contorni", "Dolci", "Bevande".	\
<i>Cliente</i>	Rappresenta l'utente, registrato o meno, che può effettuare le ordinazioni al tavolo	\
<i>Corpo del Messaggio</i>	Rappresenta il testo del messaggio inviato automaticamente dall'utente quando effettua delle interazioni, come la richiesta di un cameriere.	\
<i>Pagato</i>	È un booleano (0,1) che indica se il cliente ha pagato gli ordini immediatamente, oppure se pagherà in un diverso momento.	\
<i>Posizione</i>	È un intero che rappresenta la posizione che avranno i piatti all'interno della lista dei piatti del menù.	\
<i>Staff</i>	Rappresenta il personale del ristorante, il quale si occupa di attivare l'applicazione e sbloccarla quando il cliente ha pagato il conto.	Cameriere, Personale
<i>Testo Ordine</i>	Rappresenta il testo di tutto l'ordine con i nomi dei piatti, i prezzi, le aggiunte scelte e le note.	\
<i>Utente</i>	Rappresenta l'identità della persona che ha effettuato la registrazione.	\

**Figura 4.27.** Glossario dei termini relativo alla base di dati dell'applicazione

<b>ENTITÀ/RELATIONSHIP</b>	<b>TRADUZIONE</b>	<b>VINCOLI DI RIFERIMENTO</b>
<i>Aggiunta</i>	AGGIUNTE ( <u>ID</u> , Nome, Categoria, Prezzo)	\\
<i>Notifica</i>	MESSAGGI ( <u>TimeStamp</u> , Sessione, Tavolo, CorpoMessaggio, E-mail)	Sessione → SessioneTavolo.Sessione Tavolo → SessioneTavolo.Tavolo E-mail → Utente.Email
<i>Ordine</i>	ORDINI ( <u>TimeStamp</u> , <u>Sessione</u> , Tavolo, IDUtente, TestoOrdine, Totale, Pagato)	Sessione → SessioneTavolo.Sessione IDUtente → Utente.IDUtente
<i>Sessione Tavolo</i>	SESSIONE ( <u>Sessione</u> , Tavolo, DataInizio, DataFine)	\\
<i>Staff</i>	STAFF ( <u>ID</u> , User, Password)	\\
<i>Utente</i>	UTENTE ( <u>IDUtente</u> , E-mail, Ordini, Punti)	
<i>Voce Menù</i>	MENÙ ( <u>ID</u> , Nome, Categoria, Posizione, Ingredienti, Prezzo, LinkImmagine)	\\

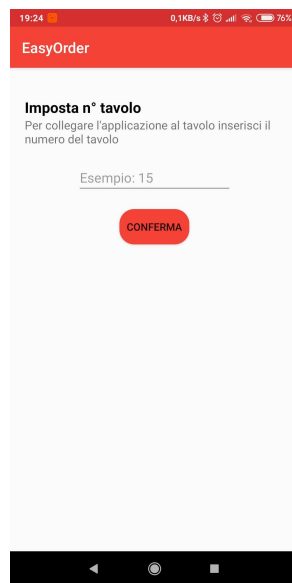
**Figura 4.28.** Traduzione verso il modello relazionale relativo alla base di dati dell'applicazione

## Implementazione

*In questo capitolo, inizialmente verranno descritte le funzionalità principali dell'applicazione per poi illustrare le scelte implementative effettuate e i componenti di terze parti utilizzati durante la progettazione.*

### 5.1 Le funzionalità dell'applicazione

La prima schermata visualizzata, dopo aver installato l'applicazione, è quella mostrata in Figura 5.1. Essa permette al cameriere di inserire il numero del tavolo corrispondente, in modo tale che i successivi ordini effettuati da quel dispositivo saranno associati al rispettivo tavolo.



**Figura 5.1.** Implementazione relativa alla pagina “Imposta Tavolo”

Una volta impostato il numero del tavolo, l'applicazione sarà configurata e potrà essere usata dall'utente. Infatti, la schermata principale visualizzata dal cliente sarà la *home*, mostrata in Figura 5.2. In essa si avrà la possibilità di visualizzare le informazioni generali del ristorante e, inoltre, si potrà chiamare un cameriere, tramite il pulsante recante la scritta *Hai bisogno di aiuto?*.

Attraverso la scheda *Profilo* (Figura 5.4 e Figura 5.19), raggiungibile attraverso la *bottom bar*, l'utente potrà gestire, in un'unica schermata, tutte le informazioni memorizzate dall'applicazione. Egli, infatti, dopo aver effettuato la registrazione (Figura 5.13) o il login (Figura 5.12), potrà visualizzare, nella sezione *I tuoi dati*, il numero dei punti a disposizione e il numero di ordini effettuati, mentre, nei due pulsanti sottostanti, può, rispettivamente, visualizzare le informazioni fornite al momento della registrazione (Figura 5.14) e la lista degli ordini che risultano ancora da pagare (Figura 5.9).

Attraverso il menù posto nella sezione *Ordina* (Figura 5.3), l'utente potrà creare il suo ordine, filtrando prima i piatti di interesse per categoria e, successivamente, selezionando la pietanza da una ricca lista (Figura 5.5).



Figura 5.2. Realizzazione della pagina “Home”

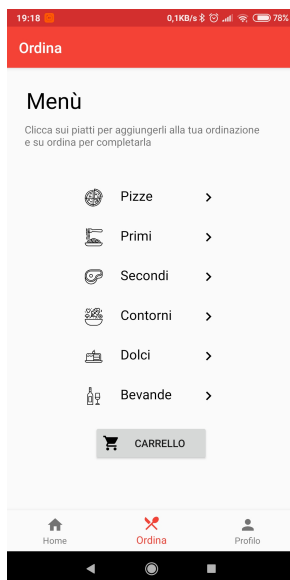


Figura 5.3. Realizzazione della pagina “Ordina”

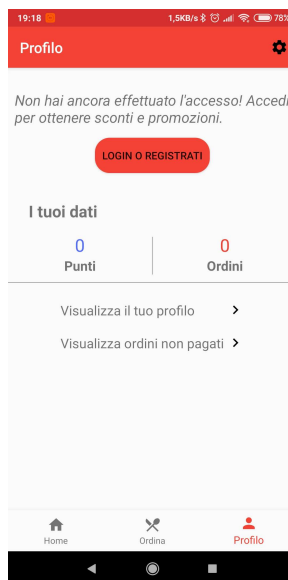


Figura 5.4. Realizzazione della pagina “Profilo”

Una volta che si sarà selezionato il piatto, l'utente potrà visualizzare gli ingredienti principali e potrà comporsi la pietanza nel modo che preferisce, tramite l'inserimento di aggiunte (Figura 5.6). Nella schermata successiva, inoltre, se l'utente ha qualche allergia o intolleranza in particolare, potrà notificarlo alla cucina inserendo delle note al piatto (Figura 5.7). Una volta decisa la pietanza da ordinare, egli potrà aggiungerla al carrello e comporne una nuova.

Le pietanze scelte saranno visibili nel carrello (Figura 5.8) e potranno essere cancellate facilmente, tramite l'apposito pulsante a forma di cestino, prima di essere





Figura 5.5. Realizzazione della pagina “Pietanza”



Figura 5.6. Realizzazione della pagina “Aggiunte”

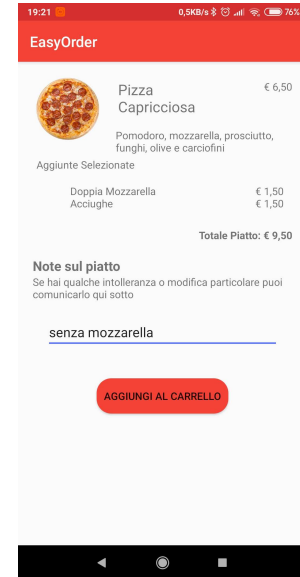


Figura 5.7. Realizzazione della pagina “Note”

inviare alla cucina.

Gli ordini effettuati potranno essere visualizzati all'interno della pagina *Comande* (Figura 5.9), raggiungibile dalla scheda *Ordina*. Essendo l'applicazione sviluppata per un ristorante, si è avuta la necessità di fornire al cliente la possibilità di effettuare più ordini prima di procedere al pagamento. Una volta che il cliente ha terminato il pasto e vuole saldare il conto, potrà cliccare sul pulsante *Paga ora* per iniziare la procedura.

L'utente avrà, quindi, la possibilità di scegliere il metodo di pagamento (Figura 5.10) e, inoltre, se registrato, potrà usufruire di particolari sconti, tramite l'utilizzo di punti guadagnati in precedenza (Figura 5.18).

Una volta deciso se pagare tramite contanti o con la carta di credito, l'applicazione rimarrà bloccata nella scheda *Login Staff* (Figura 5.11), invierà al personale la richiesta del pagamento del tavolo in questione e resterà in attesa dell'arrivo di un cameriere.

Accedendo nell'*Area Riversata* (Figura 5.15), il cameriere potrà visualizzare la pagina *Impostazioni Staff* (Figura 5.16) e avrà la possibilità di confermare il pagamento del cliente, indicando se questo ha pagato o no (Figura 5.17). Potrebbe, infatti, succedere che il cliente decida di pagare il suo conto a fine mese o che egli abbandoni insoddisfatto il locale senza pagare: anche in questo caso l'app deve essere sbloccata notificando cosa è avvenuto.

Una volta confermato il pagamento, l'applicazione verrà resettata incrementando il numero della sessione e sarà pronta per il cliente successivo. Il cameriere, attraverso l'*Area Riservata*, potrà, anche, cambiare il numero del tavolo e visualizzare con quale tavolo e con quale sessione è configurato il dispositivo.

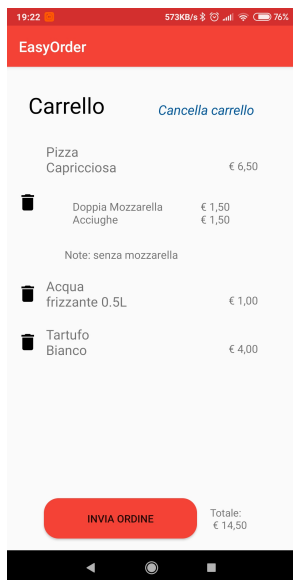


Figura 5.8. Realizzazione della pagina “Carrello”

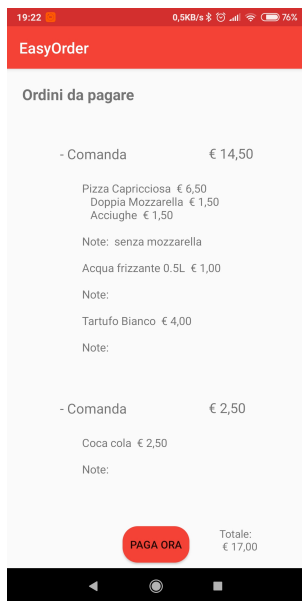


Figura 5.9. Realizzazione della pagina “Comande”

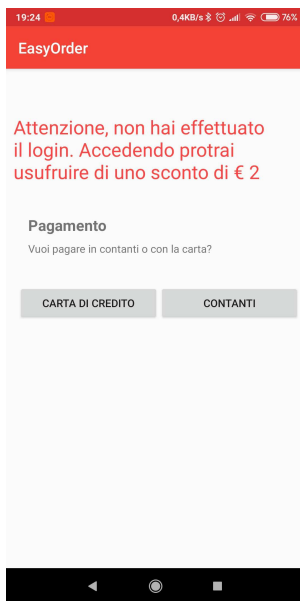


Figura 5.10. Realizzazione della pagina “Pagamento”

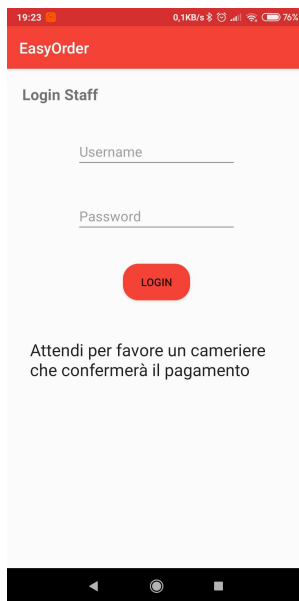


Figura 5.11. Realizzazione della pagina “Login Staff”

## 5.2 Componenti di Firebase utilizzate

Firestore è una piattaforma di Google che, grazie ai numerosi strumenti e alle API che fornisce, permette di costruire con facilità robuste applicazioni, sia web che

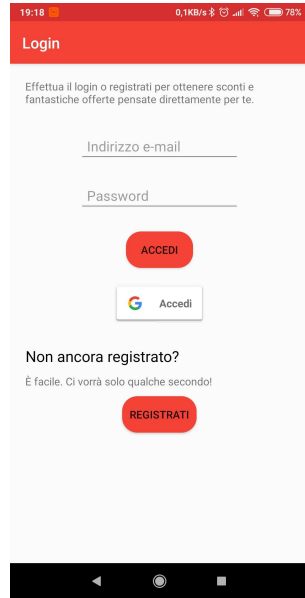


Figura 5.12. Realizzazione della pagina “Login Utente”

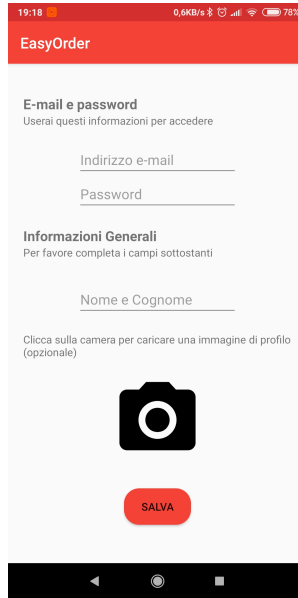


Figura 5.13. Realizzazione della pagina “Registrazione”

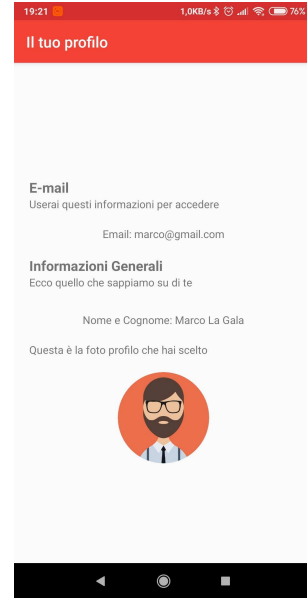


Figura 5.14. Realizzazione della pagina “Visualizza Informazioni”

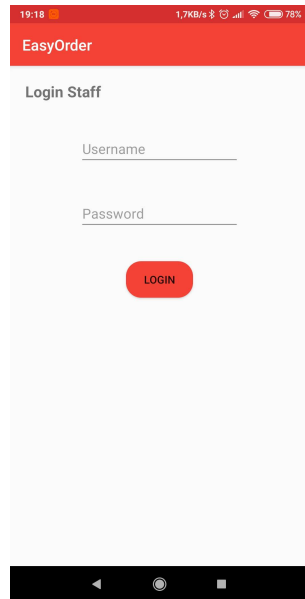


Figura 5.15. Realizzazione della pagina “Area Riservata”

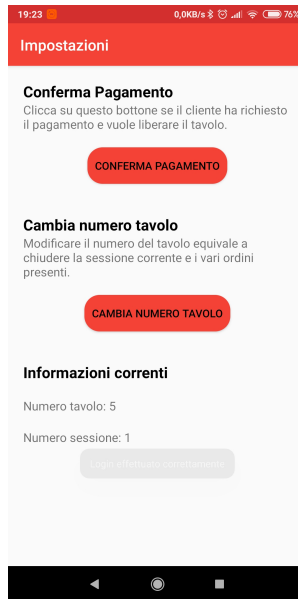


Figura 5.16. Realizzazione della pagina “Impostazioni Staff”

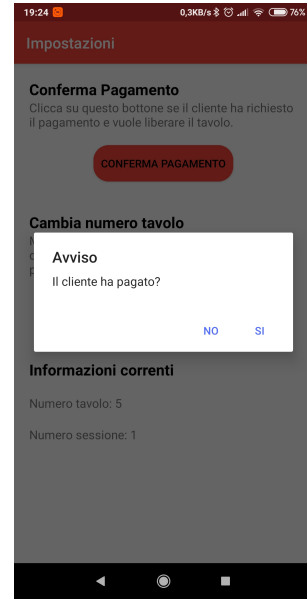


Figura 5.17. Pop-up che permette di impostare il pagamento del cliente

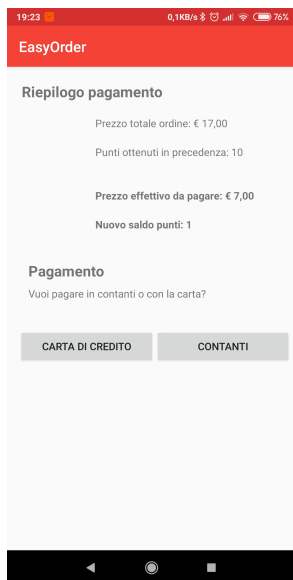


Figura 5.18. Realizzazione della pagina “Pagamento” con utente registrato

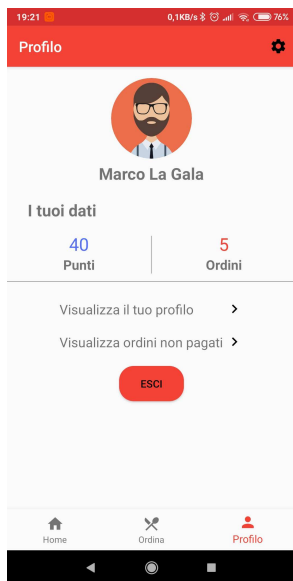


Figura 5.19. Realizzazione della pagina “Profilo” con utente registrato

mobile. Essa supporta, infatti, i principali linguaggi di programmazione, come *Swift*, *Objective-C*, *Java*, *Javascript*, *C++*, etc.

Come è possibile vedere dalla Figura 5.20, la piattaforma fornisce moltissimi strumenti, i quali coprono una vasta area di servizi che gli sviluppatori possono utilizzare direttamente, come *analytics*, *authentication*, *databases*, *configuration*, *file*

*storage, push messaging, etc.*



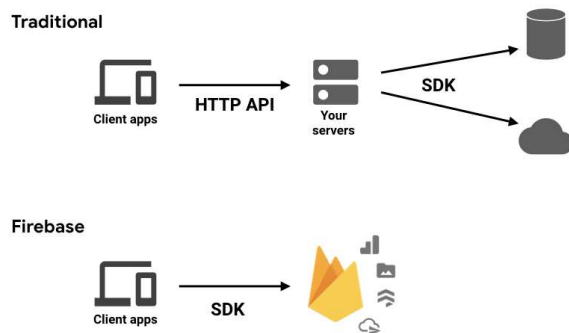
**Figura 5.20.** Elenco dei servizi e degli strumenti messi a disposizione da Firebase

Due sono i principali vantaggi che hanno determinato la scelta di questo servizio. Il primo è che, utilizzando componenti già sviluppati e mantenuti direttamente da Google, è possibile concentrarsi direttamente sull'effettivo sviluppo delle funzionalità dell'applicazione. In questo modo uno sviluppatore non deve preoccuparsi di programmare tutte le funzionalità di contorno, ma può fare affidamento su una solida e ben progettata piattaforma.

Il secondo motivo, invece, è legato al fatto che, come è possibile vedere in Figura 5.21, i vari SDK forniti da *Firebase* interagiscono direttamente con l'applicazione client, senza la necessità di stabilire alcun *middleware*, indispensabile, invece, nelle applicazioni tradizionali. Il tutto viene, quindi, ulteriormente semplificato.

### 5.2.1 Firebase Storage

*Firebase Storage* è, principalmente, un archivio di file e permette di implementare all'interno dell'app funzioni come il caricamento online di foto e video. Nel progetto sviluppato, questa componente viene utilizzata per recuperare le immagini delle varie pietanze del menù e per salvare l'immagine del profilo scelta dall'utente.



**Figura 5.21.** Schematizzazione dell'architettura tradizionale e dell'architettura con Firebase

Il Listato 5.1 mostra il metodo `uploadImageToFirebaseStorage`, contenuto nella classe `SignInActivity`, e permette di salvare online l'immagine scelta dall'utente (`uriProfileImage`).

Come primo passo abbiamo creato una variabile di tipo `StorageReference` impostando il percorso in cui l'immagine verrà salvata; essa sarà memorizzata all'interno della cartella `profilepics` e avrà come estensione `.jpg` (riga 2). Successivamente, richiamiamo il metodo `putFile` di `profileImageRef`, il quale seleziona l'immagine scelta dall'utente e la carica online. Se il caricamento è andato a buon fine, verrà eseguito il metodo `onSuccess`, il quale restituisce l'URL dell'immagine, mentre, se durante la procedura si verificano degli errori, si attiverà il metodo `onFailure`, il quale restituirà un opportuno messaggio.

```

1  private void uploadImageToFirebaseStorage() {
2      final StorageReference profileImageRef = FirebaseStorage.getInstance().getReference("profilepics/"
3      + System.currentTimeMillis() + ".jpg");
4
5      if (uriProfileImage != null) {
6
7          profileImageRef.putFile(uriProfileImage)
8              .addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
9              @Override
10             public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
11
12                 profileImageRef.getDownloadUrl().addOnCompleteListener(new OnCompleteListener<Uri>() {
13                 @Override
14                 public void onComplete(@NonNull Task<Uri> task) {
15                     profileImageURL=task.getResult().toString();
16                 }
17             });
18         }
19     })
20     .addOnFailureListener(new OnFailureListener() {
21     @Override
22     public void onFailure(@NonNull Exception e) {
23         progressBar.setVisibility(View.GONE);
24         Toast.makeText(SignInActivity.this, "Errore Caricamento "+e.getMessage(),
25         Toast.LENGTH_SHORT).show();
26     }
27     });
28 }
29
30 }

```

**Listato 5.1.** Definizione del metodo `uploadImageToFirebaseStorage` della classe `SignInActivity`

## 5.2.2 Firebase Database

*Firestore Database* consente la gestione di un *Database Realtime*, il quale permette di archiviare e sincronizzare i dati in tempo reale su tutte le applicazioni connesse. Durante lo sviluppo dell'applicazione mobile si è fatto largo uso di questo componente, in quanto ci ha permesso di memorizzare dati, come le informazioni dell'utente, i vari ordini effettuati, oppure le varie pietanze del menù. Il database di Firebase utilizza JSON per memorizzare i dati. Questo formato di dati prevede vari nodi innestati tra di loro e permette di memorizzare l'informazione tramite una coppia di elementi chiave/valore.

In Figura 5.22 vengono mostrati i principali nodi che compongono il nostro database. Si è deciso di convertire uno ad uno le varie tabelle ottenute con la progettazione logica e concettuale della Sezione 4.2.

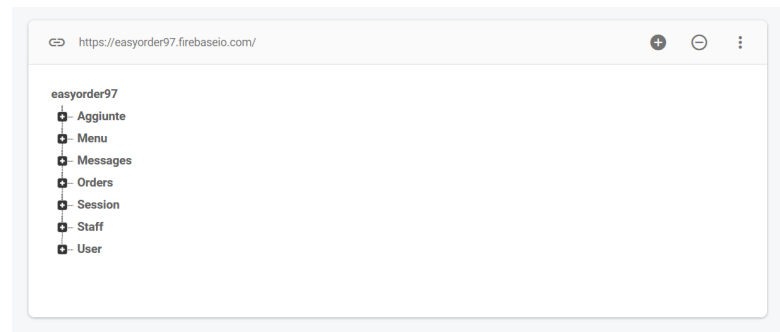


Figura 5.22. Realtime Database di Firebase

Proponiamo, di seguito, il Listato 5.2 il quale mostra la funzione `ReadMenu`. Questo metodo permette di associare un `EventListener` ad un oggetto `mReferenceMenu` di tipo `DatabaseReference`. Come è possibile vedere all'interno del costruttore, alla riga 7, `mReferenceMenu` viene referenziato soltanto al nodo chiamato *Menù*.

La funzione `onDataChange`, di conseguenza, si attiva quando i dati del nodo in questione vengono modificati; essa riceve come argomento un oggetto di tipo `DataSnapshot`, il quale contiene tutti i dati presenti all'interno del nodo *Menù*. Tramite un ciclo `for`, che ripercorre tutte le chiavi del nodo, si vanno a memorizzare le chiavi estratte all'interno di una lista di stringhe chiamata `keys`, mentre i vari valori di ogni nodo vengono memorizzati in una lista di oggetti *Menù* chiamata `menu` (riga 23).

Infine, per completezza, riportiamo nella Figura 5.23, l'esempio di un nodo di tipo *Menù*. Esso è composto da ulteriori nodi interni che rappresentano l'id del singolo piatto; all'interno di ciascuno di essi troviamo gli elementi che compongono la pietanza, come la categoria, il link dell'immagine, il nome, la posizione e il prezzo.

```

1 private DatabaseReference mReferenceMenu;
2 private FirebaseDatabase mDatabase;
3
4 public FirebaseDatabaseHelper() {
5
6     mDatabase = FirebaseDatabase.getInstance();
7     mReferenceMenu = mDatabase.getReference("Menu");

```

```

8      mReferenceAggiunte = mDatabase.getReference("Aggiunte");
9      mReferenceStaff = mDatabase.getReference("Staff");
10     mReferenceOrders = mDatabase.getReference("Orders");
11   }
12
13   public void ReadMenu(final DataStatus dataStatus){
14     mReferenceMenu.addValueEventListener(new ValueEventListener() {
15       @Override
16       public void onDataChange(@NonNull dataSnapshot) {
17         menu.clear();
18         List<String> keys = new ArrayList<>(); //save the key on the node
19         for (DataSnapshot keynode : dataSnapshot.getChildren()){ //ciclo for per elemento
20
21           keys.add(keynode.getKey()); //put inside the key array the key of the element
22           Menu elem_menu = keynode.getValue(Menu.class);
23           menu.add(elem_menu);
24         }
25         dataStatus.DataIsLoaded(menu, keys);
26       }
27     }
28     @Override
29     public void onCancelled(@NonNull DatabaseError databaseError) {
30
31     }
32   });
33 }

```

Listato 5.2. Definizione parziale della classe `FirebaseDatabaseHelper`



Figura 5.23. Esempio di nodo contenente le informazioni di un oggetto di tipo menù

### 5.2.3 Firebase Authentication

Un altro componente importante di Firebase, chiamato *Firebase Authentication*, gestisce l'autenticazione degli utenti. Esso permette, infatti, di memorizzare le informazioni degli utenti in tutta sicurezza e consente di implementare, in modo semplice, l'autenticazione tramite password, numero di telefono o tramite i *federated identity provider*, come Google, Facebook e Twitter.

Per consentire ad un utente di accedere ad un'app si devono ottenere le sue credenziali di autenticazione. Tali credenziali possono essere l'indirizzo e-mail e la password dell'utente nonché una *token OAuth* fornito dai *federated identity provider*. Successivamente, si passano tali credenziali all'SDK di autenticazione di Firebase, il quale, attraverso la sua componente di back-end, le verificherà e restituirà il risultato, ossia se i dati inseriti sono corretti o meno.



Dopo aver eseguito correttamente l'accesso, si possono ottenere ulteriori informazioni, fornite al momento della registrazione dall'utente, come l'e-mail, il nome utente, il codice identificativo, etc.

Nel Listato 5.3 viene mostrata la classe `LogInActivity`, la quale permette ad un utente di effettuare il login. Impostiamo il `setOnClickListener` sul pulsante del login (`mSignIn_btn`) in modo tale che, al click, verifichi se l'e-mail e la password sono corrette. Per fare ciò, utilizziamo il metodo `signInWithEmailAndPassword`, il quale riceve come argomenti l'e-mail e la password inserite nei campi di testo e ne verifica la correttezza. Se esse risultano corrette, viene attivato il metodo `onSuccess`, il quale riporta alla home; in caso contrario, viene attivato il metodo `onFailure`.

Assume particolare importanza il componente `mAuth`, il quale si caratterizza con l'identità dell'utente e, interrogandolo, permette di verificare, prima di tutto, se egli ha effettuato il login. Successivamente possiamo utilizzare le sue proprietà per ottenere ulteriori informazioni. Infine, per effettuare il logout, si dovrà, semplicemente, richiamare il metodo `mAuth.signOut()`, il quale resetterà il componente `mAuth` eliminando le informazioni contenute al suo interno.

```

1 public class LogInActivity extends AppCompatActivity implements View.OnClickListener {
2
3     private EditText mEmail_editTxt;
4     private EditText mPassword_editTxt;
5
6     private Button mSignIn_btn;
7
8     private DatabaseReference mDatabase;
9     private ProgressBar mProgress_bar;
10    private FirebaseAuth mAuth;
11
12    @Override
13    protected void onCreate(Bundle savedInstanceState) {
14        super.onCreate(savedInstanceState);
15        setContentView(R.layout.activity_log_in);
16        mAuth = FirebaseAuth.getInstance();
17
18        findViewById(R.id.sign_in_button).setOnClickListener(this);
19
20        mDatabase = FirebaseDatabase.getInstance().getReference();
21
22        mEmail_editTxt = findViewById(R.id.email_editText);
23        mPassword_editTxt = findViewById(R.id.password_editText);
24
25        mSignIn_btn = findViewById(R.id.sign_in_btn);
26
27        mSignIn_btn.setOnClickListener(new View.OnClickListener() {
28            @Override
29            public void onClick(View v) {
30                if (isEmpty()) return;
31                inProgress(true);
32                mAuth.signInWithEmailAndPassword(mEmail_editTxt.getText().toString(),
33                    mPassword_editTxt.getText().toString())
34                    .addOnSuccessListener(new OnSuccessListener<AuthResult>() {
35                        @Override
36                        public void onSuccess(AuthResult authResult) {
37                            Toast.makeText(LogInActivity.this, "Login effettuato con successo",
38                                Toast.LENGTH_LONG).show();
39                            Intent intent = new Intent(LogInActivity.this, MainActivity.class);
40                            intent.putExtra("FRAGMENT_ID", 2);
41                            intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
42                            startActivity(intent);
43                            finish();
44                            return;
45                        }
46                    }).addOnFailureListener(new OnFailureListener() {
47                        @Override
48                        public void onFailure(@NonNull Exception e) {
49                            inProgress(false);
50                            Toast.makeText(LogInActivity.this, "Errore " + e.getMessage(), Toast.LENGTH_LONG).show();
51                        }
52                    });
53            }
54        });
55    }
56
57    //controlla se i campi e-mail e password sono vuoti. In tal caso mostra un errore
58    private boolean isEmpty() {
59        if (TextUtils.isEmpty((mEmail_editTxt.getText().toString()))) {
60            mEmail_editTxt.setError("DATO RICHIESTO!");

```

```

61     return true;
62   }
63   if (TextUtils.isEmpty(mPassword_editTxt.getText().toString())) {
64     mPassword_editTxt.setError("DATO RICHIESTO!");
65     return true;
66   }
67   return false;
68 }
69 }

```

**Listato 5.3.** Definizione parziale della classe LoginActivity

## 5.3 Implementazione delle componenti dell'applicazione

Di seguito riportiamo alcune soluzioni adottate durante lo sviluppo della nostra applicazione per raggiungere gli obiettivi proposti.

### 5.3.1 Blocco dell'app con Shared Preferences

La prima funzionalità che mostriamo riguarda il blocco dell'applicazione dopo che il cliente ha richiesto il pagamento.

Il cliente, infatti, dopo aver richiesto il pagamento, non deve più avere la possibilità di effettuare ulteriori ordini; per tale ragione, si è deciso di implementare un meccanismo che bloccasse completamente l'applicazione sulla schermata di login dell'*Area riservata*, accessibile solo al personale del ristorante.

Utilizziamo, così, un componente di Android chiamato *Shared Preferences*, il quale permette di garantire la persistenza dei dati attraverso l'utilizzo di coppie chiave/valore. Abbiamo memorizzato, infatti, una chiave di nome `BlockPref`, che rappresenta lo stato in cui si trova l'applicazione; la variabile assume valore 0 se l'applicazione non è bloccata mentre vale 1 in caso contrario.

Nel Listato 5.4 viene mostrato il metodo `AppBlock` della classe `MainActivity`, il quale verifica, all'apertura dell'app, se il valore di `BlockPref` è 1. In caso affermativo, e quindi se l'app è bloccata, rimandiamo direttamente all'activity `LoginStaff`. In questo modo, anche se l'applicazione viene riavviata, rimane comunque bloccata.

```

1  public void AppBlock(){
2  if ((BlockPref.getInt("BlockPref", -1))==1){
3
4      Log.i("BlockPref", "onCreate:HOME APP BLOCCATA");
5      Intent k = new Intent(this, LoginStaff.class);
6      startActivity(k);
7
8  }
9  }

```

**Listato 5.4.** Definizione del metodo AppBlock della classe MainActivity

Una volta che il cameriere è arrivato al tavolo può confermare il pagamento del cliente e, di conseguenza, può eliminare il blocco all'app in modo tale che quest'ultima possa essere utilizzata dai clienti successivi.

Creiamo, quindi, un oggetto `BlockPref` di tipo `SharedPreferences` e, utilizzando la proprietà di nome `Editor`, modifichiamo il valore della chiave `BlockPref`

impostandolo a 0, eliminando, di conseguenza, il blocco dell'app (riga 19 - Listato 5.5).

```

1  public void Finisci(){
2      // Update TimeFine with current time
3      Log.i("DB", "Chiudo sessione precedente ");
4      SimpleDateFormat en = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
5      String en_date = en.format(new Date());
6
7      mDatabase.child("Session").child("Session: "+Session).child("timeFine").setValue(en_date);
8
9      // Find the new id session (search max into firebase), add session to firebase and memorize the id
10     //into sharedpref
11     NuovaSession();
12
13     // Make the Cat empty
14     ManagerCarrello.ResettaCarrello();
15
16     //Eliminate user mAuth
17     mAuth.signOut();
18
19     //Eliminate the Block of the app
20     Log.i("BlockPref", "Tolgo il blocco");
21     SharedPreferences BlockPref = getSharedPreferences("BlockPref", 0);
22     SharedPreferences.Editor editor2 = BlockPref.edit();
23     editor2.putInt("BlockPref", 0);
24     editor2.apply();
25
26     Intent intent = new Intent(Settings.this, MainActivity.class);
27
28     // Serve per cancellare lo stack delle activity
29     intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
30
31     startActivity(intent);
32     finish();
33 }

```

**Listato 5.5.** Definizione del metodo `Finisci`

Abbiamo, infine, modificato il metodo `onBackPressed` della classe `Login`, in modo tale che, se viene impostato il blocco dell'app, l'utente non potrà tornare alle activity precedenti cliccando sul tasto `Back` del dispositivo (Listato 5.6).

```

1  @Override
2  public void onBackPressed() {
3
4      SharedPreferences BlockPref = getSharedPreferences("BlockPref", 0);
5      if ((BlockPref.getInt("BlockPref", -1))!=1){
6          Intent intent = new Intent(this, LoginStaff.class);
7          intent.putExtra("chiamatautente", 1);
8          startActivity(intent);
9
10         Log.i("BlockBACK", "onClick: NO ");
11     }
12     else{
13         Log.i("BlockBACK", "onClick: SI ");
14         super.onBackPressed();
15     }
16 }

```

**Listato 5.6.** Definizione del metodo `onBackPressed` della classe `Login`

### 5.3.2 Sistema fidelizzazione utenti

Si è deciso di creare un sistema di fidelizzazione degli utenti in modo tale che questi ultimi, invogliati dagli sconti, siano più propensi a ritornare al ristorante. Per fare ciò, abbiamo deciso di attribuire un punto, del valore di un euro, ogni 5 euro di spesa. L'utente, quindi, prima di procedere al pagamento dell'ordine, avrà uno sconto del valore dei punti ottenuti in precedenza.

Nel Listato 5.7 viene mostrato il codice per l'attribuzione dei punti. Dopo aver salvato su una variabile interna `mPoints` il numero dei punti in possesso dall'utente,

calcoliamo il nuovo totale dell'ordine che l'utente dovrà pagare dopo aver sottratto lo sconto all'ordine originario. Prima di tutto (riga 12), si verifica se il numero di punti in possesso è maggiore del totale dell'ordine. Se ciò risulta vero, l'utente non dovrà pagare nulla per l'ordine ed avrà, come nuovo saldo punti, quello precedente meno il totale dell'ordine. Al contrario, se risulta minore, avrà come nuovo totale dell'ordine quello precedente meno i punti ottenuti in precedenza, e come punti, la nuova spesa diviso per cinque.

```

1 public void onDataChange(DataSnapshot dataSnapshot) {
2     // Get Post object and use the values to update the UI
3     Users data = dataSnapshot.getValue(Users.class);
4
5     assert data != null;
6     Log.i("DATI", "onDataChange: DATI: " + data.getPoints());
7     mPoints = data.getPoints();
8
9     mTotalOrder_txt.setText(getString(R.string.payment_totalOrder, mTotalOrder));
10    mPoints_txt.setText(getString(R.string.payment_oldpoints, mPoints));
11
12    if(mPoints>mTotalOrder) {
13        mNewTotalOrder = 0; //visto che ha più punti del totale da pagare, scallamo tutti i punti
14        //possibili e lui paga 0 euro
15        mNewPoints = (int) (mPoints - mTotalOrder); //visto che paga 0 euro non avrà nuovi punti e
16        //quindi calcolo i punti rimasti
17    }
18    else {
19        mNewTotalOrder=mTotalOrder-mPoints;
20        mNewPoints = (int) (mNewTotalOrder/5);
21    }
22
23    mNewTotalOrder_txt.setText(getString(R.string.payment_newtotal, mNewTotalOrder));
24    mNewPoints_txt.setText(getString(R.string.payment_newPoints, mNewPoints));
25
26    setmNewPoints(mNewPoints);
27 }

```

**Listato 5.7.** Definizione del metodo `onDataChange` della classe `ProcediAlPagamento` per l'attribuzione dei punti

### 5.3.3 Sessione Utente

Si è deciso di implementare il concetto di sessione, in modo tale da identificare univocamente il cliente. L'applicazione, infatti, memorizzerà sul database di Firebase le varie sessioni con l'orario di inizio e di fine. Si è deciso di far iniziare la sessione quando si imposta un nuovo numero del tavolo o quando termina la sessione precedente, dopo che il cameriere ha confermato il pagamento.

Nel Listato 5.8 viene mostrata la funzione `NuovaSessione` della classe `Imposta Tavolo`, la quale memorizza una nuova sessione nel database. Prima di tutto, salviamo tutte le sessioni che sono presenti online, e verifichiamo il caso in cui non ci sia alcuna sessione. Se, infatti, il `dataSnapshot` prelevato è nullo (riga 9), allora la sessione che dobbiamo creare è la numero 1.

Memorizziamo sul dispositivo, sotto forma di `SharedPreferences`, una chiave di nome `SessPref` del valore di 1 e creiamo l'oggetto `newsession` di tipo `Session`, che rappresenta la nuova sessione, inserendo al proprio interno il numero della sessione, la data di inizio e, come data di fine, la stringa *sessione in corso*, che sarà, poi, modificata alla chiusura della sessione. Infine memorizziamo, all'interno del database, l'oggetto creato.

```

1 public void NuovaSessione(){
2     DatabaseReference ref = FirebaseDatabase.getInstance().getReference().child("Session");
3     ref.addListenerForSingleValueEvent(

```

```

4         new ValueEventListener() {
5
6             @Override
7             public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
8
9                 if (dataSnapshot.getValue() == null) { //se è la prima volta il nodo sessione non esiste,
10                    // lo creiamo con una sessione 1
11                    SharedPreferences SessPref = getSharedPreferences("SessPref", 0);
12                    SharedPreferences.Editor editor1 = SessPref.edit();
13                    editor1.putInt("SessPref", 1);
14                    editor1.apply();
15
16                    SimpleDateFormat en = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
17                    String en_date = en.format(new Date());
18                    int mSession = 1;
19                    Session newsession = new Session(mSession, en_date, "sessione in corso");
20                    FirebaseDatabase.getInstance().getReference().child("Session").child("Sessione:
21                    "+mSession).setValue(newsession);
22                }
23                else {
24                    collectSessionNumbers((Map<String, Object>) dataSnapshot.getValue());
25                }
26            }
27
28            @Override
29            public void onCancelled(@NonNull DatabaseError databaseError) {
30
31            }
32        });
33
34    }

```

**Listato 5.8.** Definizione del metodo NuovaSessione della classe ImpostaTavolo

Se, invece, il `dataSnapshot` ha elementi al proprio interno, richiamiamo la funzione `collectSessionNumbers` la quale, a differenza di quella precedentemente illustrata, ricercherà il massimo tra i numeri di sessione prelevati, in quanto il numero della sessione dovrà essere quello successivo al massimo trovato, ossia `massimo+1` (Listato 5.9).

```

1     private void collectSessionNumbers(Map<String, Object> users) {
2         ArrayList<Long> sessionNumbers = new ArrayList<>();
3         SharedPreferences SessPref = getSharedPreferences("SessPref", 0);
4         //iterate through each user, ignoring their UID
5         for (Map.Entry<String, Object> entry : users.entrySet()){
6
7             //Get user map
8             Map singleUser = (Map) entry.getValue();
9             //Get phone field and append to list
10            sessionNumbers.add((Long) singleUser.get("id"));
11        }
12
13        Long max = Collections.max(sessionNumbers);
14        int massimo = max.intValue();
15
16        SharedPreferences.Editor editor1 = SessPref.edit();
17        editor1.putInt("SessPref", massimo+1);
18        editor1.apply();
19
20        // Memorizzo la sessione su Firebase
21        SimpleDateFormat en = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
22        String en_date = en.format(new Date());
23        Session newsession = new Session(massimo+1, en_date, "sessione in corso");
24        int numSessione = massimo+1;
25        FirebaseDatabase.getInstance().getReference().child("Session").child("Sessione:
26        "+numSessione).setValue(newsession);
27    }

```

**Listato 5.9.** Definizione del metodo `collectSessionNumbers` della classe ImpostaTavolo



## Discussione in merito al lavoro svolto

*In questo capitolo affronteremo un'analisi critica del lavoro svolto nella presente tesi, valutando i punti di forza e di debolezza dell'applicativo realizzato. Confronteremo, inoltre, le analogie e le differenze con altri sistemi simili che sono presenti sul mercato.*

### 6.1 Punti di forza del sistema realizzato

Dopo un'attenta analisi critica dell'applicativo, si è deciso di elencare i vari punti di forza del sistema realizzato.

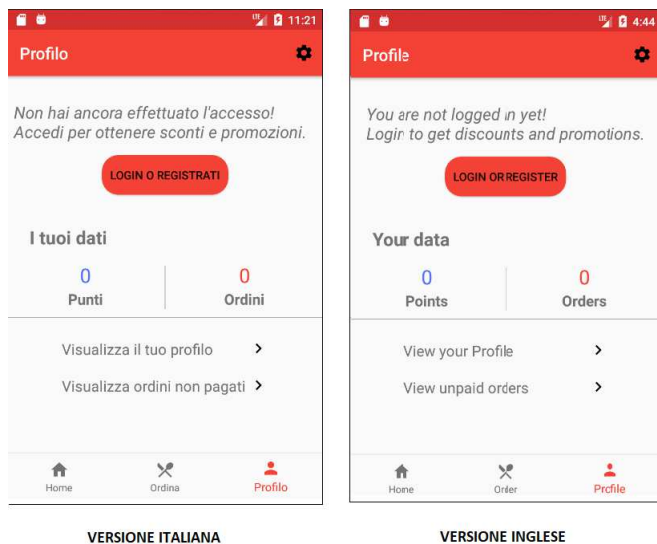
Si può evidenziare, infatti, che l'applicazione memorizza e utilizza dati contenuti in un database online. Grazie ai componenti messi a disposizione da *Firebase*, infatti, siamo riusciti a creare un base di dati dove memorizzare le informazioni del menù e dell'utente. In questo modo, con pochi e semplici click, si potranno modificare le pietanze presenti sul menù o aggiungerne di nuove e il sistema continuerà a funzionare correttamente senza dover distribuire una nuova versione dell'app.

Si è deciso, inoltre, di attuare una strategia simile anche per la sezione delle aggiunte del singolo piatto. Esistono, infatti, delle categorie, come la *carne* o i *dolci*, dove non sono previste aggiunte e, di conseguenza, l'utente non visionerà la scheda relativa. Il personale del ristorante, però, nel momento in cui decide di creare delle aggiunte anche per quelle categorie di piatti, non dovrà modificare l'applicativo, ma, semplicemente, inserire nel database le aggiunte, collegandole alla suddetta categoria. In questo modo l'applicazione, notando la presenza di elementi per quella categoria, mostrerà in automatico al cliente la sezione per la personalizzazione del piatto.

Un ulteriore aspetto importante dell'applicazione riguarda l'implementazione del profilo dell'utente. Egli, infatti, può registrarsi con e-mail e password, oppure tramite Google, e, per completare la registrazione, deve fornire ulteriori dati, come, per esempio, l'immagine del profilo, scelta dal rullino fotografico.

Nella scheda "Profilo", inoltre, il cliente potrà visualizzare tutti gli ordini effettuati e non ancora saldati, anche relativi a sessioni precedenti a quella attuale. In questo modo, l'utente che, nelle precedenti sessioni, non ha saldato il conto, potrà pagarlo in quelle successive.

Per quanto riguarda la questione della lingua, essendo il ristorante frequentato anche da turisti, si è deciso di implementare, oltre all'italiano, anche la lingua inglese. In questo modo, cambiando la lingua del dispositivo, le varie stringhe presenti all'interno dell'applicazione saranno visualizzate in inglese. È possibile vedere un esempio nella Figura 6.1, dove viene mostrata la scheda "Profilo" nelle due lingue supportate.



**Figura 6.1.** Versione della scheda "Profilo" nelle due lingue disponibili

L'ultimo elemento che vogliamo porre in evidenza è il fatto che il sistema funziona ed è stabile anche quando più dispositivi effettuano ordinazioni contemporaneamente. Essendo un'applicazione installata all'interno dei dispositivi posizionati sui tavoli, essa deve essere in grado di funzionare anche quando un numero significativo di utenti la sta utilizzando. Inoltre, l'applicazione sarà completamente bloccata e, di conseguenza, inutilizzabile, dopo che il cliente ha richiesto il pagamento, in quanto l'utente non dovrà effettuare ulteriori ordini. Il cliente, quindi, anche riavviando l'applicazione o premendo il tasto indietro del dispositivo, non riuscirà a aggirare il blocco.

## 6.2 Punti di debolezza del sistema realizzato

Durante l'analisi critica, oltre ai punti di forza dell'applicativo, si sono riscontrate delle debolezze. Queste potranno essere migliorate se si deciderà di continuare il progetto in altri ambiti.

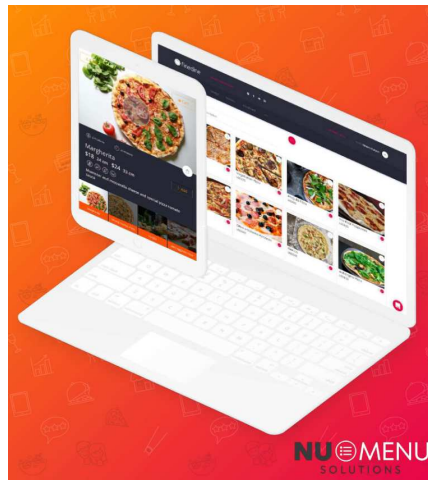
La criticità più importante è legata al fatto che non è prevista un'app o un sito web, utilizzabile dal cameriere, per visualizzare i messaggi inviati o gli ordini. Egli, infatti, dovrà accedere al database di Firebase per visionarli. Ciò non è dovuto a



problemi implementativi, ma, semplicemente, si è ritenuto opportuno concentrare lo sviluppo soltanto sull'app utilizzata dal cliente.

Un ulteriore punto di debolezza, che ci sentiamo di evidenziare, è l'impossibilità di modificare il menù direttamente nell'applicazione, o attraverso una piattaforma web (Figura 6.2). Non abbiamo previsto, infatti, uno strumento che permetta al personale di modificare e/o aggiungere pietanze al menù. Il cameriere, di conseguenza, per effettuare delle modifiche, dovrà accedere alla piattaforma fornita da Firebase e modificare direttamente la base di dati. Ciò potrebbe risultare ostico e difficile per il ristoratore e, inoltre, potrebbe portare all'introduzione di errori che rovinerebbero la persistenza dei dati all'interno del database.

Inoltre, non abbiamo implementato un sistema di pagamento *in-app* per gli ordini ma, per semplicità, abbiamo previsto che fosse esclusivamente il cameriere ad effettuarlo, tramite contanti o un terminale POS del locale. Si è deciso di effettuare questa scelta principalmente perché implementare un sistema di pagamento sicuro e a prova di manomissioni può risultare difficile e dispendioso.



**Figura 6.2.** Esempio di piattaforma web dove il personale può modificare il menù

In ultimo, l'applicazione potrebbe essere ampliata con ulteriori funzionalità che ne migliorino l'esperienza dell'utente e semplifichino il lavoro del personale. Si potrebbe, infatti, implementare le *notifiche push*, permettendo al ristorante di pubblicizzare particolari iniziative, oppure si potrebbe aggiungere al menù una sezione con le offerte del giorno o con le pietanze in sconto.

### 6.3 Analogie con altri sistemi simili esistenti sul mercato

Esistono diverse soluzioni sul mercato simili all'applicativo realizzato come, per esempio, *Nu-Menu* (<https://www.nu-menu.co.za/>) oppure *Menu* (<https://menu>)

.app/en/). Le caratteristiche in comune con l'applicazione progettata sono molteplici. La prima è la possibilità di avere un menù interattivo in cui l'utente può visionare gli ingredienti e le foto dei singoli piatti e aggiungerli direttamente al carrello. Come è possibile vedere dalla Figura 6.3, questi sistemi consentono, inoltre, di poter modificare la singola pietanza con l'inserimento di aggiunte.

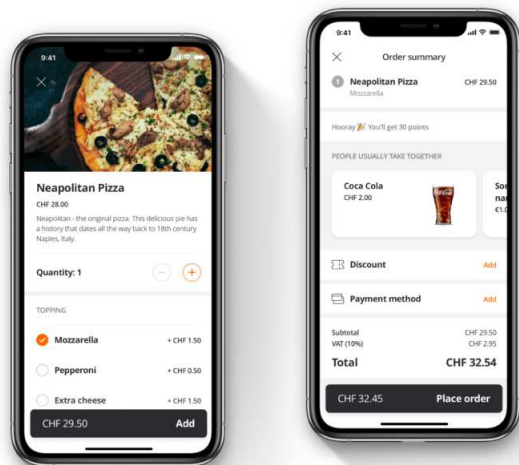


Figura 6.3. Sezione relativa al menù dell'app *iMenu*

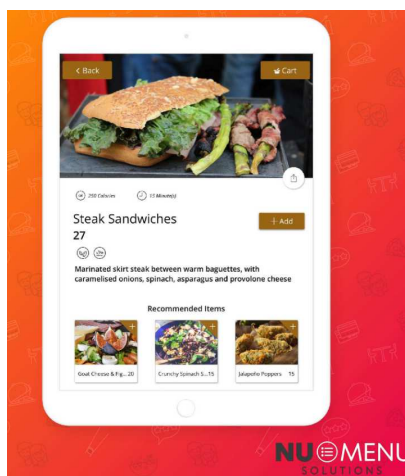
Inoltre, nella Figura 6.4, viene mostrata la scheda relativa ad una pietanza dell'applicazione *Nu-Menu*. È possibile notare che questi sistemi sfruttano la risoluzione del dispositivo per mostrare immagini del prodotto in alta definizione e che forniscono all'interno della scheda ulteriori informazioni, come gli ingredienti, le calorie o i piatti correlati a quello scelto.

Un'ulteriore caratteristica in comune tra i vari sistemi è quella di implementare un profilo utente e progettare funzionalità per fidelizzare il più possibile i clienti. I vari applicativi in commercio, infatti, permettono di effettuare particolari sconti selezionati in base al tipo di utenza registrata, oppure, tramite l'inserimento di *codici coupon*. Essi offrono, infine, anche la possibilità di visualizzare pubblicità all'interno dell'app per sponsorizzare particolari piatti o novità.

## 6.4 Differenze rispetto ad altri sistemi simili esistenti sul mercato

Analizzando le alternative presenti sul mercato, è possibile distinguere alcune differenze rispetto all'applicativo realizzato.

La prima differenza notata è che il nostro applicativo ha bisogno costantemente della connessione ad Internet per funzionare correttamente. Ciò è dovuto principal-



**Figura 6.4.** Scheda relativa ad un prodotto dell'app *Nu-Menu*

mente al fatto che l'app deve comunicare con il database online per recuperare i dati da mostrare all'interno.

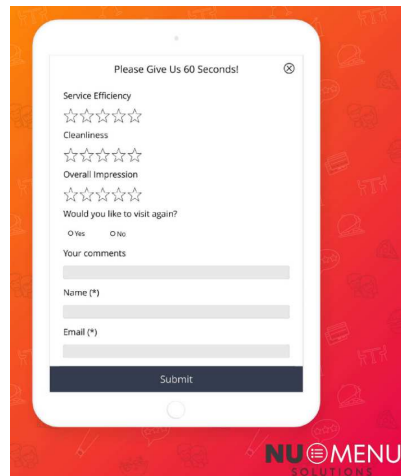
I sistemi in commercio, invece, permettono di utilizzare l'app per alcune funzionalità, come la visualizzazione del menù, senza la necessità di avere una connessione ad Internet costante. Il sistema, però, ha bisogno di essere connesso per inviare gli ordini effettuati o per visualizzare promozioni speciali.

Inoltre, quasi nessuna delle applicazioni esistenti permette di inviare direttamente messaggi al cameriere come, per esempio, per richiedere aiuto. Questa funzionalità, invece, potrebbe essere molto utile quando il ristorante si trova a servire clienti che non sono abituati ad utilizzare questo tipo di tecnologia e hanno bisogno dell'aiuto del personale.

Una funzionalità interessante, invece, che non è presente nell'applicativo, ma risulta di grande utilità, è la possibilità di recensire il servizio di ristorazione o il singolo piatto scelto. Ciò permette al ristorante di comprendere quanto gli utenti siano soddisfatti del locale e delle sue pietanze (Figura 6.5).

Navigando tra i vari sistemi esistenti, si è riscontrato, inoltre, che ognuno adotta una soluzione diversa per impostare il numero del tavolo. Partendo dal presupposto che questa informazione è di notevole importanza per fare in modo che il cameriere consegni il piatto al tavolo corretto, si è deciso che, nel nostro applicativo, questo venga impostato direttamente dal personale. In questo modo, si evitano errori, sia volontari che involontari, nella gestione dei tavoli. Altri sistemi, invece, come quello di *WMenu* ([wmenu.com](http://wmenu.com)), mostrato in Figura 6.6, sfruttano un *codice QR*, posizionato sul tavolo, per configurare l'applicazione; altre soluzioni permettono direttamente al cliente di digitarlo.

Infine, un'ulteriore differenza risiede nel fatto che non abbiamo considerato di sviluppare un'applicazione per il sistema operativo IOS. Ormai, quando si progettano sistemi di questo genere bisogna fare in modo che l'applicazione sia compatibile con quanti più dispositivi possibili, in modo tale da poter essere usata da eventuali dispositivi già presenti nel locale.

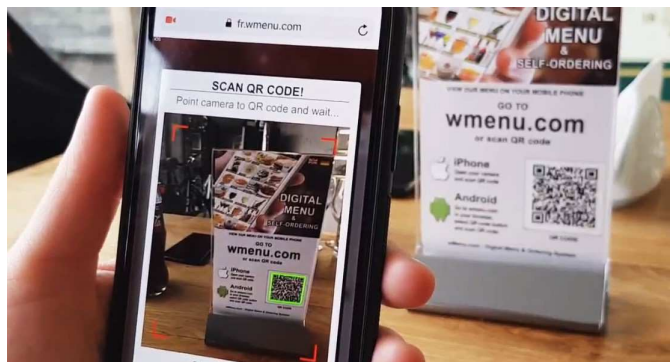


The image shows a white tablet displaying a review form for 'Nu-Menu Solutions'. The form is titled 'Please Give Us 60 Seconds!' and includes the following sections:

- Service Efficiency:** Five empty star icons.
- Cleanliness:** Five empty star icons.
- Overall Impression:** Five empty star icons.
- Would you like to visit again?:** Two radio buttons labeled 'Yes' and 'No'.
- Your comments:** A text input field.
- Name (\*):** A text input field.
- Email (\*):** A text input field.
- Submit:** A dark blue button at the bottom.

The background of the tablet is a red gradient with faint icons of various food items. The 'NU-MENU SOLUTIONS' logo is visible in the bottom right corner of the tablet screen.

**Figura 6.5.** Sezione relativa alla recensione del servizio dell'applicazione *Nu-Menu*



**Figura 6.6.** Funzionalità del codice QR nell'applicazione *Wmenu*

## Conclusioni

Nella presente tesi è stata progettata e realizzata un'applicazione mobile che permettesse agli utenti di effettuare autonomamente ordinazioni al ristorante. Tale applicazione, infatti, implementa un menù interattivo in cui l'utente può scegliere e personalizzare le varie pietanze da ordinare. L'applicazione, inoltre, è stata sviluppata in modo tale che tutte le informazioni, salvate online, possano essere recuperate facilmente da altri sistemi, o direttamente dal personale del locale.

Inizialmente, per comprendere meglio il mondo mobile, abbiamo approfondito l'evoluzione dei telefoni cellulari, concentrandoci sull'introduzione del concetto di app e sulla nascita dalla figura dello sviluppatore mobile. Successivamente, dopo aver scelto di sviluppare l'applicazione per il sistema operativo Android, essendo questo il sistema operativo più diffuso, abbiamo studiato l'ecosistema del "robotto verde" nella sua interezza, per comprendere quali strumenti e quali funzionalità avevano a disposizione gli sviluppatori. Abbiamo appreso, così, l'architettura del sistema operativo e le varie componenti da cui è formata un'applicazione Android.

Si è poi proceduto ad un'analisi dei requisiti, la quale ci ha permesso di delineare il progetto nella sua interezza ed elencare le caratteristiche che il sistema sviluppato avrebbe dovuto implementare. Successivamente, siamo passati alla progettazione, inizialmente dell'interfaccia dell'applicazione, tramite *Mockup* e *Flow Chart*, per poi dedicarci alla progettazione sia concettuale che logica della base di dati. Infine, nella fase relativa all'implementazione, abbiamo realizzato le funzionalità principali dell'applicativo avvalendoci, anche, del supporto prezioso di Firebase. In ultimo, abbiamo effettuato vari test per raffinare l'applicazione e renderla priva di errori.

Analizzando il progetto sviluppato è possibile individuare aspetti che potrebbero essere migliorati o ulteriori funzionalità implementabili. Si potrebbe, infatti, per quanto riguarda il sistema di fidelizzazione degli utenti, fornire promozioni particolari in base ai giorni della settimana, oppure utilizzare le *notifiche push* per pubblicizzare eventi particolari. Ulteriore mancanza, inoltre, è un'applicazione, o un sito web, dove il personale del ristorante possa vedere gli ordini e le notifiche inviate dal cliente.



---

## Ringraziamenti

Desidero, innanzitutto, ringraziare tutti coloro che mi hanno aiutato nella realizzazione del progetto e nella stesura di questa tesi di laurea. Ringrazio il mio relatore, il Professore Domenico Ursino, per la sua competenza e per la sua disponibilità e rapidità nel curare e correggere la tesi, capitolo dopo capitolo.

Vorrei ringraziare la mia famiglia, che ha sempre creduto in me e mi ha sempre sostenuto, sia moralmente che economicamente, durante questo percorso di studi.

Ringrazio, inoltre, tutti i miei amici e tutti i miei compagni di corso, che mi hanno aiutato in questi anni e, in particolare, il mio collega Giacomo Vitali, con il quale ho condiviso questo progetto.





---

## Riferimenti bibliografici

1. Steve Jobs Introducing The iPhone At MacWorld 2007. <https://www.youtube.com/watch?v=x7qPAY9JqE4&t=61s>, 2010.
2. Jobs' original vision for the iPhone: No third-party native apps. <https://www.9to5mac.com/2011/10/21/jobs-original-vision-for-the-iphone-no-third-party-native-apps/>, 2011.
3. Original Android Prototype Revealed During Google, Oracle Trial. [https://www.pcworld.com/article/254539/original\\_android\\_prototype\\_revealed\\_during\\_google\\_oracle\\_trial.html](https://www.pcworld.com/article/254539/original_android_prototype_revealed_during_google_oracle_trial.html), 2012.
4. Android founder: We aimed to make a camera OS. <https://www.pcworld.com/article/2034723/android-founder-we-aimed-to-make-a-camera-os.html>, 2013.
5. Guida Android. <https://www.html.it/guide/guida-android/>, 2014.
6. App: Native, Ibride o Web? [https://www.evoluzioniweb.it/IT/Mobile\\_App\\_Native\\_Ibride\\_Web](https://www.evoluzioniweb.it/IT/Mobile_App_Native_Ibride_Web), 2015.
7. Tutorial, Com'è fatta l'architettura di Android ? <http://www.quickgo.it/tutorial-come-fatta-larchitettura-di-android/#sthash.QHv0wdvo.dpbs>, 2015.
8. Una breve storia dei telefoni cellulari dal 1973 a oggi. <https://www.stelladoradus.it/la-storia-dei-telefoni-cellulari/>, 2015.
9. 10 anni fa Steve Jobs presentava il primo iPhone. <http://www.rainews.it/dl/rainews/media/10-anni-anni-fa-Steve-Jobs-presentava-il-primi-iPhone-2f341920-e109-44e3-b396-58ad629943a4.html#foto-1>, 2017.
10. Creare app Android: struttura delle applicazioni e AVD. [https://www.ilsoftware.it/articoli.asp?tag=Creare-app-Android-struttura-delle-applicazioni-e-AVD\\_16528](https://www.ilsoftware.it/articoli.asp?tag=Creare-app-Android-struttura-delle-applicazioni-e-AVD_16528), 2017.
11. The History of Mobile Phone Technology. <https://www.redorbit.com/reference/the-history-of-mobile-phone-technology/>, 2018.
12. What is Firebase? The complete story, abridged. <https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0>, 2018.
13. Android. [https://en.wikipedia.org/wiki/Android\\_\(operating\\_system\)](https://en.wikipedia.org/wiki/Android_(operating_system)), 2020.
14. App Store. [https://it.wikipedia.org/wiki/App\\_Store](https://it.wikipedia.org/wiki/App_Store), 2020.
15. Distribution dashboard. <https://developer.android.com/about/dashboards>, 2020.
16. Motorola DynaTAC. [https://en.wikipedia.org/wiki/Motorola\\_DynaTAC#cite\\_note-7](https://en.wikipedia.org/wiki/Motorola_DynaTAC#cite_note-7), 2020.
17. Operating System Market Share. <https://netmarketshare.com/operating-system-market-share.aspx>, 2020.

18. Report: Top 100 App Store publishers earning over 60% more than Android counterparts. [https://www.gsmarena.com/report\\_top\\_100\\_app\\_store\\_publishers\\_earning\\_over\\_60\\_more\\_than\\_android\\_competitors-news-37691.php](https://www.gsmarena.com/report_top_100_app_store_publishers_earning_over_60_more_than_android_competitors-news-37691.php), 2020.
19. M. Burton and D. Felker. *Android App Development For Dummies*. John Wiley Sons Inc, 2015.
20. E. Cisotti and M. Giannino. *Android: Guida completa*. Edizioni LSWR, 2015.
21. F. Darwin. *Android Cookbook: Problems and Solutions for Android Developers*. O'Reilly Media, 2012.
22. T. Hagos. *Learn Android Studio 3*. Apress, 2018.
23. N. Smyth. *Android Studio 3.0 Development Essentials: Android 8 Edition*. Createspace Independent Pub, 2017.