Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Sviluppo di un'app mobile per la gestione dei pasti aziendali con controllo automatico delle presenze

Tesi di laurea

Relatore

Prof. Ombretta Gaggi

Laureando

Erica Cavaliere - 2013450

Anno Accademico 2022-2023

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage della laureanda Erica Cavaliere presso l'azienda RiskApp S.r.l.
È stato richiesto lo sviluppo di un'applicazione mobile per i pasti aziendali con un sistema di autenticazione degli utenti e che permetta di controllare la loro presenza a pranzo.
Viene riportato nello specifico gli obbiettivi richiesti e le funzioni implementate, descrivendo lo sviluppo del progetto affidato.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine alla Prof. Ombretta Gaggi, relatrice della mia tesi, per l'aiuto, il sostegno, la disponibilità e la professionalità che mi ha fornito durante la stesura del lavoro.

Ringrazio il team di RiskApp per avermi accolto, in particolare Luca, il mio tutor aziendale, per l'aiuto fornitomi durante lo stage.

Desidero ringraziare con affetto la mia famiglia per il sostegno in questi anni di studio, in particolare i miei genitori per essermi stati vicini e per il loro aiuto nei momenti di dubbio e di difficoltà.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme durante questo percorso, in particolare Gabriel, il mio ragazzo, per l'aiuto e il sostegno che mi ha sempre dato.

Granze, Novembre 2023 Erica Cavaliere

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

RiskApp S.r.l. (Figura 1.1) è un'azienda con sede a Conselve (PD) che si occupa di sviluppo software per il mondo assicurativo.
È stata fondata nel 2016 e il suo *core business* è lo sviluppo e il mantenimento dell'omonima applicazione, che viene costantemente aggiornata ed estesa per garantire un prodotto che possa rispondere ad ogni esigenza.
Il principale punto di forza di questa piattaforma è quello di stimare le possibili perdite economiche di un'impresa attraverso un algoritmo proprietario che, anche attraverso l'uso dell'intelligenza artificiale, valuta il rischio raccogliendo e combinando una moltitudine di dati da diverse fonti.
Il personale aziendale lavora costantemente per migliorare i propri servizi, ragionando sui possibili problemi che l'utente e le aziende possono andare incontro, fanno riunioni e call per capire come migliorare e ampliare la piattaforma, tutto svolto in un clima di calma e rispetto tra colleghi.

Figura 1.1: Logo dell'azienda RiskApp

1.2 L'idea

Per poter gestire le spese per i pasti, che preparano in azienda, è stato scelto di sviluppare un'app mobile che permetta di monitorare i versamenti degli utenti, scegliere il piatto del giorno da un menu condiviso e monitorare la *cassa comune*[g].
Deve essere gestita l'autenticazione di ogni utente, dividendo tra utente semplice e utente amministratore e permettere il controllo delle presenze in azienda durante i pranzi.
Ogni utente potrà aggiungere un piatto nel menu, proporre il pasto del giorno, monitorare la sua *quota stornata*[g] e la *cassa comune*, indicare le spese effettuate e modificare i dati personali.
L'amministratore potrà anche gestire le presenze e le spese effettuate dagli stagisti.
L'applicazione dovrà essere sviluppata con *Flutter*[g], *Dart*[g] e *Firebase*[g].

1.3 Obiettivi

Di seguito sono riportati gli obiettivi concordati per lo svolgimento del progetto, ovvero tutti i requisiti che il prodotto finale dovrà rispettare.
Si farà riferimento ai requisiti secondo le seguenti notazioni:

- O per i requisiti obbligatori, vincolanti in quanto obiettivi primari richiesti dal committente;
- D per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- F per i requisiti facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Le sigle precedentemente indicate saranno seguite da una coppia sequenziale di numeri, identificativo del requisito.

Tabella 1.1: Tabella degli obiettivi

Obiettivo	Descrizione
O01	Accesso tramite credenziali
O02	Pannello di controllo degli utenti
O03	Modifica o aggiunta delle spese
O04	Monitoraggio della cassa comune
O05	Controllo presenza in azienda di una persona durante i pranzi
O06	Impostare il menu del giorno
O07	Scelta di un piatto dal menu
D01	Integrazione di ChatGPT per consigliare alcune ricette
F01	Test a livello di frontend

1.4 Organizzazione del testo

Il secondo capitolo descrive in che modo è stato creato il prodotto desiderato, quale metodo di sviluppo è stato utilizzato e quali sono le tecnologie adottate per lavorare al progetto.

Il terzo capitolo approfondisce i requisiti riportati nella sezione 1.3 Obiettivi con una analisi dettagliata di quanto è stato richiesto.

Il quarto capitolo spiega come è stata progettata l'applicazione e come è stato poi strutturato il codice del prodotto.

Nel quinto capitolo viene fatto un resoconto di tutti gli obiettivi raggiunti, riportando poi delle valutazioni personali sul lavoro svolto e sul prodotto finale.

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

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

Capitolo 2

Processi e metodologie

In questo capitolo viene spiegato il Material Design che sta alla base della progettazione dell'app, viene poi riportato il metodo di lavoro utilizzato e infine le tecnologie adottate per lo sviluppo del progetto.

2.1 Material Design

Alla base dell'applicazione, è stato scelto di seguire il Material Design (Figura 2.1) sviluppato da Google, che si concentra su un maggiore uso di *layout* basati su una griglia, animazioni, transizioni ed effetti di profondità come l'illuminazione e le ombre. Si tratta di una serie di regole ideate per consentire una buona *User Experience (UX)*[g] e definire una *User Interface (UI)*[g] per l'utente da implementare in ambiente Web, Android e in *Flutter*.
Viene annunciato per la prima volta da Google il 25 giugno del 2014 durante il Google I/O, una conferenza organizzata annualmente da Google a Mountain View, in California.

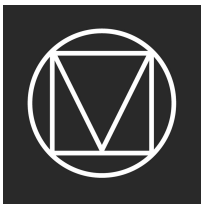

Figura 2.1: Logo del Material Design di Google

Venne rinnovato nel 2018 con il Material Design 2, anche chiamato Google Material Theme, introducendo un maggiore utilizzo di angoli arrotondati, spazi bianchi e icone colorate, infine viene rinnovato nel 2021 con il Material Design 3, oppure Material You, introducendo l'uso di tasti più grandi e maggiore uso delle animazioni.
Oggi viene ancora utilizzato il Material Design 3 ed è stato seguito per lo sviluppo dell'app dei pranzi.

Per consentire l'uso dei propri prodotti software a più utenti possibili, il Material

Design segue le regole del *Web Content Accessibility Guidelines (WCAG)*[g], mettendo alla base di ogni progetto l'accessibilità, creando così dei prodotti inclusivi, cioè usabili da tutti i tipi di utenti, anche con disabilità, consentendo a ciascuno un'esperienza fluida e semplice da usare.

I *layout* devono essere studiati in modo da guidare l'utente nella navigazione della pagina e devono essere dinamici, in modo che le pagine si adattino ad ogni tipo di schermo.
Vengono indicate delle regole precise su come devono essere impostate le *componenti*[g], come devono essere raggruppate, lo spazio che deve esserci e tanti altri piccoli ma importanti dettagli che lo sviluppatore deve considerare per permettere all'utente di orientarsi su qualsiasi dispositivo.
Anche *Flutter* offre una guida sulle *componenti* che mette a disposizione per lo sviluppatore e che sono state ideate per rispettare le regole di Material Design appena descritte.

2.2 Metodo di lavoro

Durante lo stage, RiskApp contava circa dieci dipendenti e ognuno era incaricato di sviluppare e mantenere una parte della loro piattaforma, confrontandosi tra loro ogni giorno per capire come continuare a lavorare.
Il loro metodo di lavoro si avvicina a un metodo Agile, più precisamente ad uno SCRUM, utilizzato anche per lo sviluppo del progetto di stage.

Il Manifesto per lo sviluppo Agile (*Manifesto Agile*. URL: https://agilemanifesto.org/iso/it/manifesto.html) è composto da dodici principi fondamentali che descrivono il modo in cui deve lavorare il team, permettendo possibili cambiamenti in corso d'opera e mettendo al primo posto il cliente, rilasciando varie versioni del prodotto funzionante dopo brevi periodi e privilegiando le comunicazioni faccia a faccia.
Lo SCRUM è un *framework* di gestione dei progetti Agile che mira a cinque valori fondamentali: impegno, focus, apertura, rispetto e coraggio.
Questo *framework* ha acquisito negli ultimi anni una straordinaria popolarità nel mondo dell'informatica grazie ai vantaggi offerti, come maggiore collaborazione con l'utente finale, il suo contributo al miglioramento continuo e la superiore gestione dei rischi.
L'idea di fondo consiste nel suddividere i periodi di lavoro in *sprint* di durata fissata, caratterizzati da un insieme di obiettivi da realizzare (*sprint backlog*).

Per lo sviluppo del progetto di stage, ogni giorno veniva riportato quanto era stato fatto e veniva mostrato il funzionamento, raccogliendo possibili idee per migliorare o modificare l'app.
Se in corso d'opera venivano incontrate eventuali problematiche sullo sviluppo, si ragionava su come affrontare o modificare il prodotto per risolvere questi problemi, permettendo così di soddisfare ogni esigenza degli utenti finali, in questo caso per soddisfare le esigenze dei dipendenti dell'azienda.

2.3 Tecnologie

2.3.1 Flutter

Flutter (Figura 2.2) è un progetto open-source di Google il cui vantaggio principale è la generazione di applicazioni multipiattaforma a partire da un unico codice sorgente. Permette quindi allo sviluppatore di concentrarsi sul prodotto da realizzare senza dover preferire un sistema operativo mobile ad un altro.
Per questo motivo è stato scelto di utilizzare Flutter come *framework* principale, dato che il prodotto finale deve funzionare sia per dispositivi Android sia per dispositivi iOS.

Figura 2.2: Logo di Flutter

2.3.2 Dart

Il linguaggio sul quale si basa Flutter è Dart (Figura 2.3), nato con l'intento di sostituire JavaScript come protagonista nello sviluppo delle applicazioni.
Tra i suoi pregi si elencano il compilatore JIT, migliore gestione della sicurezza, la velocità e la maggiore scalabilità.
Il paradigma principale è l'orientamento agli oggetti, una sua particolarità è data dalla sua attenzione alla *null safety*, per la quale nessun valore può essere nullo a meno che questa possibilità non sia esplicitamente dichiarata.

Figura 2.3: Logo di Dart

2.3.3 Firebase

Firebase (Figura 2.3) è una piattaforma *open-source* per la creazione di applicazioni per dispositivi mobili e web sviluppata da Google.
Firebase sfrutta l'infrastruttura di Google e il suo cloud per fornire una suite di strumenti per scrivere, analizzare e mantenere applicazioni *cross-platform*.
Infatti offre funzionalità come analisi, database (usando strutture noSQL), messaggistica e segnalazione di arresti anomali per la gestione di applicazioni web, iOS e Android.
Per lo sviluppo dell'app sono stati utilizzati:

- Firebase Autentication, per permettere la registrazione e l'autenticazione di un utente tramite mail e password;

- Cloud Firestore, per la gestione del database.

Figura 2.4: Logo di Firebase

2.3.4 Figma

Figma (Figura 2.5) è un software per la progettazione di *User Interface(UI)*.
Permette infatti di realizzare prototipi delle interfacce, detti anche *mockup*, che permettono di illustrare il risultato finale che si desidera ottenere.
Questo strumento è stato utilizzato per mostrare e concordare l'interfaccia dell'app con il tutor aziendale, prima della fase di codifica.

Figura 2.5: Logo di Figma

2.3.5 Android Studio

Android Studio (Figura 2.6) è un *Integrated Development Environment (IDE)*[g] adibito per la creazione di applicazioni Android e mette a disposizione dei simulatori virtuali di uno o più cellulari con il sistema operativo di Google.
Il progetto è stato sviluppato interamente con l'uso di questo *IDE* ed è stato utilizzato il simulatore virtuale di Google Pixel 7 con sistema operativo Android 13 per testare la *build*[g] dell'app.

Figura 2.6: Logo di Android Studio

2.3.6 Xcode

Xcode (Figura 2.7) è un *IDE* completamente sviluppato e mantenuto da Apple, contenente una suite di strumenti utili allo sviluppo di software per i sistemi macOS, iOS, iPadOS, watchOS e tvOS.
Per poter testare la *build* del progetto, è stato utilizzato il simulatore virtuale di iPhone 15 con sistema operativo iOS 17, messo a disposizione da questo software.

Figura 2.7: Logo di Xcode

2.3.7 GitHub

GitHub (Figura 2.8) è una piattaforma di *hosting* per ospitare codice all'interno di repository basato sul software Git.
Fornisce agli sviluppatori strumenti per migliorare e mantenere il codice come:

- *features* utilizzabili da linea di comando;
- gestione delle *pull request* e *code review*;
- strumenti per l'*issue tracking*.

La codebase della piattaforma RiskApp è suddivisa in varie repository su GitHub.
Per questo progetto, l'azienda ha riservato una repository apposita per permettermi di lavorare in autonomia al codice.

Figura 2.8: Logo di GitHub

2.3.8 Slack

Slack (Figura 2.9) è un applicazione multipiattaforma per la messaggistica istantanea tra membri di un gruppo di lavoro.
Una delle funzioni di Slack è la possibilità di organizzare la comunicazione del team attraverso canali specifici, canali che possono essere accessibili a tutto il team o solo ad alcuni membri.
È possibile inoltre comunicare con il team anche attraverso chat individuali private o chat con due o più membri.
Questo software è stato utilizzato per comunicare con il tutor aziendale da remoto e per condividere materiale.

Figura 2.9: Logo di Slack

Capitolo 3

Analisi dei requisiti

Di seguito viene riportata l'Analisi dei Requisiti del prodotto software, partendo dai casi d'uso e poi a seguire il tracciamento dei requisiti concordati con il proponente.

3.1 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi.
I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo Unified Modeling Language (UML) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.
Per convenzione i casi d'uso saranno classificati con questo codice:

UC[codice padre](.[Codice figlio])

Dove UC indica *Use Case* e i due codici sono:

- **Codice padre** è il codice identificativo numerico del dato caso d'uso;
- **Codice figlio** è il codice identificativo di un eventuale sotto caso d'uso.

3.1.1 Attori

Gli attori principali che andranno ad interagire con l'applicazione sono i seguenti:

- **Utente non autenticato**
- **Utente**
- **Amministratore**

Nell'analisi è stato identificato un quarto attore, ovvero **ChatGPT**, in quanto tra i requisiti desiderabili era prevista l'integrazione dello stesso per consigliare le ricette, ma alla fine dello stage questa integrazione non è stata svolta, preferendo dare priorità ai requisiti obbligatori e al funzionamento vero e proprio dell'applicazione.

3.1.2 Diagrammi e descrizione

Di seguito sono riportati i casi d'uso nel dettaglio, analizzando per ogni caso gli attori principali, la descrizione, la precondizione e la postcondizione; gli ultimi due riportano lo stato dell'applicazione, o lo stato di un attore, rispettivamente prima e dopo l'esecuzione del caso d'uso studiato.

Gli *use case* sono stati divisi in base alle azioni che devono essere svolte in ogni schermata dell'app (Home, Spese, Menu, Utente e Impostazioni), viene fatta eccezione per gli *use case* riguardanti il Primo Accesso e a ChatGPT perchè possono essere gestiti in schermate distinte o integrate nelle schermate principali.
Sono stati analizzati in modo distinto anche i sottocasi d'uso; per gli *use case* padre si elenca la generalizzazione dei propri sotto casi.

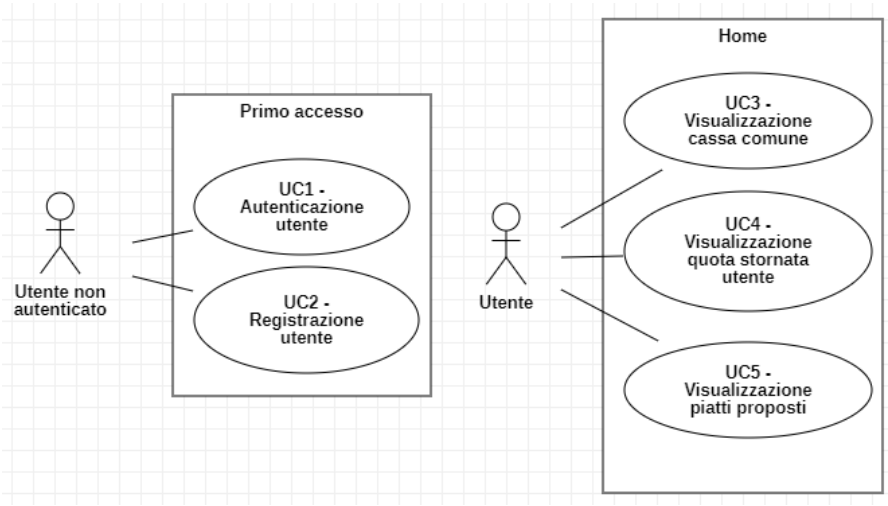

Figura 3.1: Use Case - Primo accesso e Home

UC1: Autenticazione utente - Figura 3.1

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non ha effettuato l'autenticazione.

Descrizione: L'utente inserisce la propria mail e la propria password per effettuare l'accesso.

Postcondizioni: L'utente è stato autenticato.

UC2: Registrazione utente - Figura 3.1

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non è registrato nel database.

Descrizione: L'utente inserisce i propri dati per registrarsi nel database.

Postcondizioni: L'utente è registrato nel database.

UC3: Visualizzazione *cassa comune* - Figura 3.1

Attori Principali: Utente.

Precondizioni: La *cassa comune* non è visibile.

Descrizione: L'utente entra nell'app per visualizzare la *cassa comune*.

Postcondizioni: L'utente visualizza la *cassa comune*.

UC4: Visualizzazione *quota stornata* utente - Figura 3.1

Attori Principali: Utente.

Precondizioni: La *quota stornata* dell'utente non è visibile.

Descrizione: L'utente entra nell'app per visualizzare la propria *quota stornata*.

Postcondizioni: L'utente visualizza la propria *quota stornata*.

UC5: Visualizzazione piatti proposti - Figura 3.1

Attori Principali: Utente.

Precondizioni: La lista dei piatti proposti del giorno non è visibile.

Descrizione: L'utente entra nell'app per visualizzare i piatti proposti del giorno.

Postcondizioni: L'utente visualizza la lista dei piatti proposti del giorno.

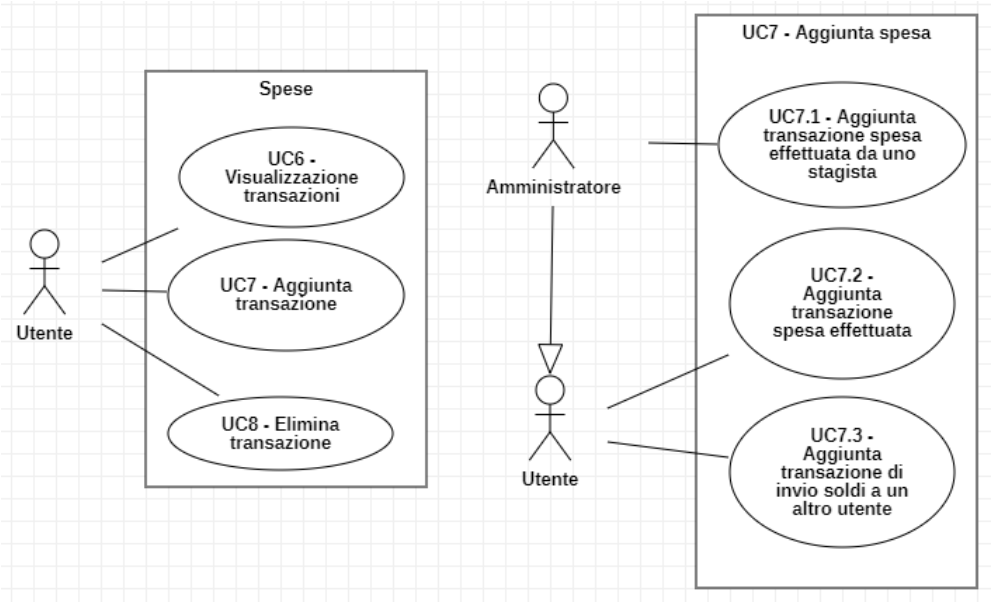

Figura 3.2: Use Case - Spese e UC7

UC6: Visualizzazione transazioni - Figura 3.2

Attori Principali: Utente.

Precondizioni: La lista delle transazioni non è visibile.

Descrizione: L'utente entra nell'app per visualizzare la lista delle transazioni.

Postcondizioni: L'utente visualizza la lista delle transazioni.

UC7: Aggiunta transazione - Figura 3.2

Attori Principali: Utente.

Precondizioni: La transazione non è presente nel database e non è visibile nella lista delle transazioni.

Descrizione: L'utente inserisce i dati della transazione interessata e la salva nel database.

Postcondizioni: La transazione è presente nel database e visibile nella lista delle transazioni.

Generalizzazioni:

- UC7.1 - Aggiunta transazione spesa effettuata da uno stagista
- UC7.2 - Aggiunta transazione spesa effettuata
- UC7.3 - Aggiunta transazione di invio soldi a un altro utente

UC7.1: Aggiunta transazione spesa effettuata da uno stagista - Figura 3.2

Attori Principali: Amministratore.

Precondizioni: La spesa effettuata da uno stagista non è presente nel database e non è visibile nella lista delle transazioni.

Descrizione: L'amministratore inserisce i dati della spesa effettuata da uno stagista e la salva nel database.

Postcondizioni: La spesa effettuata da uno stagista è presente nel database e visibile nella lista delle transazioni.

UC7.2: Aggiunta transazione spesa effettuata - Figura 3.2

Attori Principali: Utente.

Precondizioni: La spesa effettuata dall'utente non è presente nel database e non è visibile nella lista delle transazioni.

Descrizione: L'utente inserisce i dati della spesa effettuata e la salva nel database.

Postcondizioni: La spesa dell'utente è presente nel database e visibile nella lista delle transazioni.

UC7.3: Aggiunta transazione di invio soldi a un altro utente - Figura 3.2

Attori Principali: Utente.

Precondizioni: L'invio dei soldi tra due utenti non è presente nel database e non è visibile nella lista delle transazioni.

Descrizione: L'utente inserisce i dati dell'invio dei soldi a un altro utente e lo salva nel database.

Postcondizioni: L'invio dei soldi tra due utenti è presente nel database e visibile nella lista delle transazioni.

UC8: Elimina transazione - Figura 3.2

Attori Principali: Utente.

Precondizioni: La transazione è presente nel database e visibile nella lista delle transazioni.

Descrizione: L'utente elimina la transazione interessata dall'app e viene eliminata dal database.

Postcondizioni: La transazione non è presente nel database e non è visibile nella lista delle transazioni.

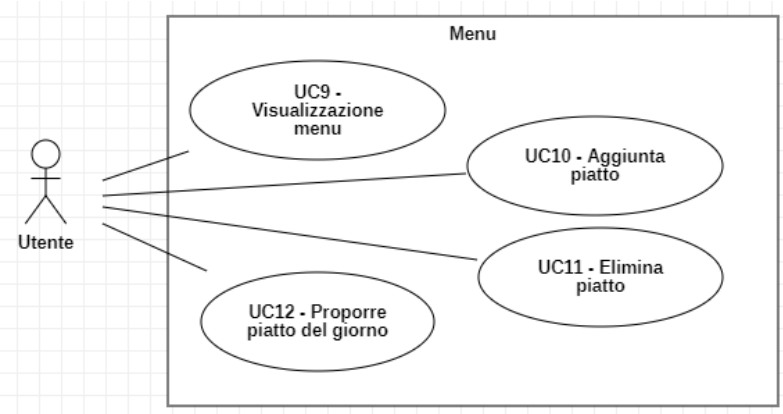

Figura 3.3: Use Case - Menu

UC9: Visualizzazione menu - Figura 3.3

Attori Principali: Utente.

Precondizioni: Il menu non è visibile.

Descrizione: L'utente entra nell'app per visualizzare il menu.

Postcondizioni: L'utente visualizza la lista dei piatti presenti nel menu.

UC10: Aggiunta piatto - Figura 3.3

Attori Principali: Utente.

Precondizioni: Il piatto non è presente nel database e non è visibile dal menu.

Descrizione: L'utente inserisce i dati del piatto e lo salva nel database.

Postcondizioni: Il piatto è presente nel database e visibile dal menu.

UC11: Elimina piatto - Figura 3.3

Attori Principali: Utente.

Precondizioni: Il piatto è presente nel database e visibile dal menu.

Descrizione: L'utente elimina il piatto interessato dall'app e viene eliminato dal database.

Postcondizioni: Il piatto non è presente nel database e non è visibile dal menu.

UC12: Proporre piatto del giorno - Figura 3.3

Attori Principali: Utente.

Precondizioni: Il piatto non è indicato come piatto proposto del giorno.

Descrizione: L'utente propone il piatto interessato come piatto del giorno.

Postcondizioni: Il piatto è indicato come piatto proposto del giorno.

UC13: Visualizzazione *quota pasto*[g] - Figura 3.4

Attori Principali: Amministratore.

Precondizioni: La *quota pasto* non è visibile.

Descrizione: L'amministratore entra nell'app per visualizzare la *quota pasto*.

Postcondizioni: L'amministratore visualizza la *quota pasto*.

UC14: Modifica *quota pasto* - Figura 3.4

Attori Principali: Amministratore.

Precondizioni: La *quota pasto* è salvata nel database con il vecchio valore.

Descrizione: L'amministratore modifica la *quota pasto* con il nuovo valore.

Postcondizioni: La *quota pasto* è salvata nel database con il nuovo valore.

UC15: Visualizzazione *quota stornata* stagisti - Figura 3.4

Attori Principali: Amministratore.

Precondizioni: La *quota stornata* dei stagisti non è visibile.

Descrizione: L'amministratore entra nell'app per visualizzare la *quota stornata* dei stagisti.

Postcondizioni: L'amministratore visualizza la *quota stornata* dei stagisti.

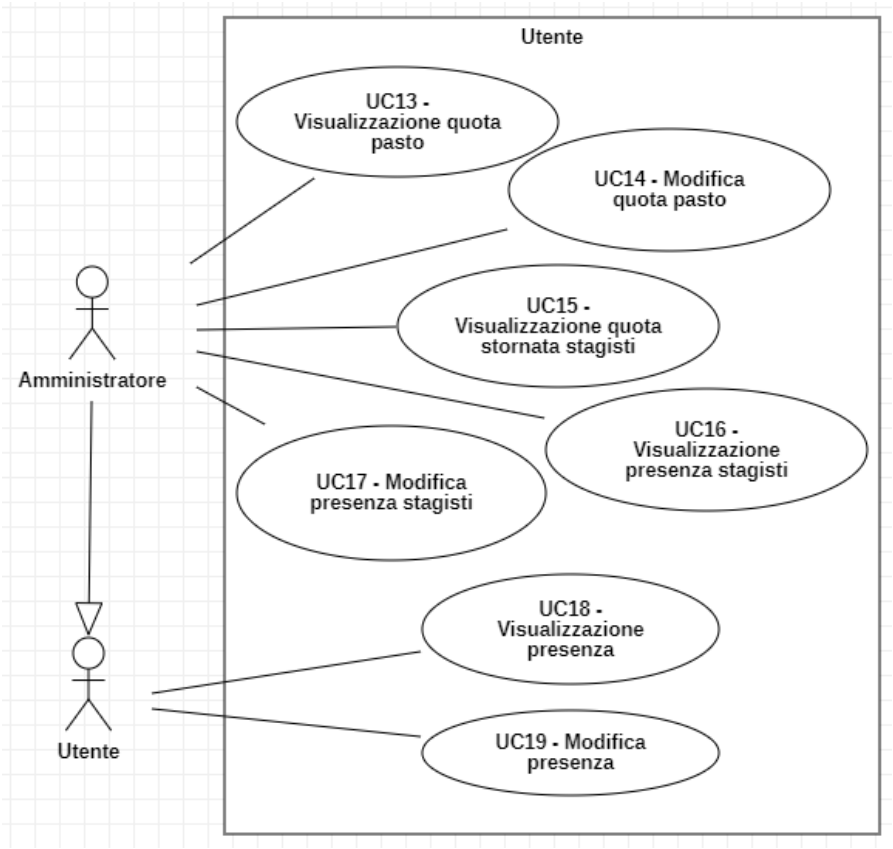

Figura 3.4: Use Case - Utente

UC16: Visualizzazione presenza stagisti - Figura 3.4

Attori Principali: Amministratore.

Precondizioni: La lista delle presenze dei stagisti non è visibile.

Descrizione: L'amministratore entra nell'app per visualizzare la lista delle presenze dei stagisti.

Postcondizioni: L'amministratore visualizza la lista delle presenze dei stagisti.

UC17: Modifica presenza stagisti - Figura 3.4

Attori Principali: Amministratore.

Precondizioni: La lista delle presenze dei stagisti è salvata nel database con i vecchi valori.

Descrizione: L'amministratore modifica la lista delle presenze dei stagisti con i nuovi valori.

Postcondizioni: La lista delle presenze dei stagisti è salvata nel database con i nuovi valori.

UC18: Visualizzazione presenza - Figura 3.4

Attori Principali: Utente.

Precondizioni: La lista delle presenze dell'utente non è visibile.

Descrizione: L'utente entra nell'app per visualizzare la lista delle proprie presenze.

Postcondizioni: L'utente visualizza la lista delle proprie presenze.

UC19: Modifica presenza - Figura 3.4

Attori Principali: Utente.

Precondizioni: La lista delle presenze dell'utente è salvata nel database con i vecchi valori.

Descrizione: L'utente modifica la lista delle proprie presenze con i nuovi valori.

Postcondizioni: La lista delle presenze dell'utente è salvata nel database con i nuovi valori.

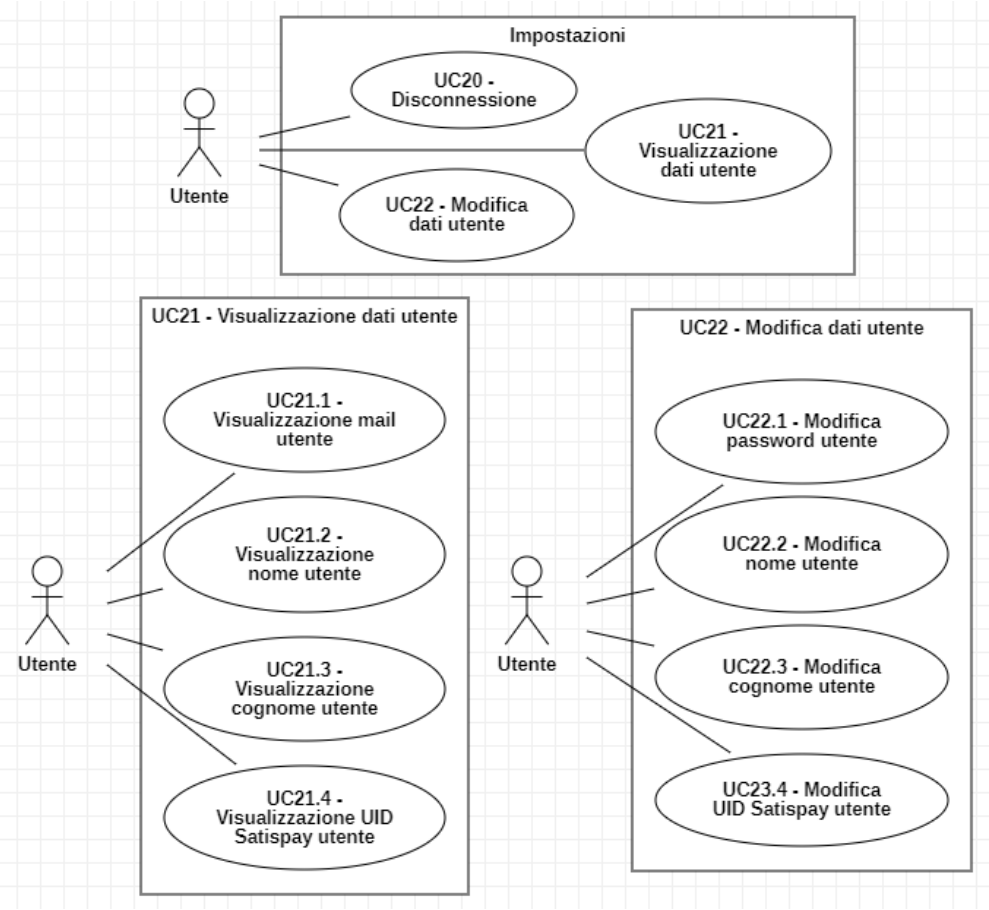

Figura 3.5: Use Case - Impostazioni, UC21 e UC22

UC20: Disconnessione - Figura 3.5

Attori Principali: Utente.
Precondizioni: L'utente è autenticato nell'app.
Descrizione: L'utente si disconnette dalla sessione corrente.
Postcondizioni: L'utente non è autenticato.

UC21: Visualizzazione dati utente - Figura 3.5

Attori Principali: Utente.
Precondizioni: I dati dell'utente non sono visibili.
Descrizione: L'utente entra nell'app per visualizzare i propri dati.
Postcondizioni: L'utente visualizza i propri dati.

Generalizzazioni:

- UC21.1 - Visualizzazione mail utente
- UC21.2 - Visualizzazione nome utente
- UC21.3 - Visualizzazione cognome utente
- UC21.4 - Visualizzazione UID Satispay utente

UC21.1: Visualizzazione mail utente - Figura 3.5

Attori Principali: Utente.
Precondizioni: La mail dell'utente non è visibile.
Descrizione: L'utente entra nell'app per visualizzare la propria mail.
Postcondizioni: L'utente visualizza la propria mail.

UC21.2: Visualizzazione nome utente - Figura 3.5

Attori Principali: Utente.
Precondizioni: Il nome dell'utente non è visibile.
Descrizione: L'utente entra nell'app per visualizzare il proprio nome.
Postcondizioni: L'utente visualizza il proprio nome.

UC21.3: Visualizzazione cognome utente - Figura 3.5

Attori Principali: Utente.
Precondizioni: Il cognome dell'utente non è visibile.
Descrizione: L'utente entra nell'app per visualizzare il proprio cognome.
Postcondizioni: L'utente visualizza il proprio cognome.

UC21.4: Visualizzazione UID Satispay utente - Figura 3.5

Attori Principali: Utente.
Precondizioni: L'UID Satispay dell'utente non è visibile.
Descrizione: L'utente entra nell'app per visualizzare il proprio UID Satispay.
Postcondizioni: L'utente visualizza il proprio UID Satispay.

UC22: Modifica dati utente - Figura 3.5

Attori Principali: Utente.
Precondizioni: I dati dell'utente sono salvati nel database con i vecchi valori.
Descrizione: L'utente modifica i propri dati con i nuovi valori.
Postcondizioni: I dati dell'utente sono salvati nel database con i nuovi valori.

Generalizzazioni:

- UC22.1 - Modifica password utente
- UC22.2 - Modifica nome utente
- UC22.3 - Modifica cognome utente
- UC22.4 - Modifica UID Satispay utente

UC22.1: Modifica password utente - Figura 3.5

Attori Principali: Utente.
Precondizioni: La password dell'utente è salvata nel database con il vecchio valore.
Descrizione: L'utente modifica la propria password con il nuovo valore.
Postcondizioni: La password dell'utente è salvata nel database con il nuovo valore.

UC22.2: Modifica nome utente - Figura 3.5

Attori Principali: Utente.
Precondizioni: Il nome dell'utente è salvato nel database con il vecchio valore.
Descrizione: L'utente modifica il proprio nome con il nuovo valore.
Postcondizioni: Il nome dell'utente è salvato nel database con il nuovo valore.

UC22.3: Modifica cognome utente - Figura 3.5

Attori Principali: Utente.
Precondizioni: Il cognome dell'utente è salvato nel database con il vecchio valore.
Descrizione: L'utente modifica il proprio cognome con il nuovo valore.
Postcondizioni: Il cognome dell'utente è salvato nel database con il nuovo valore.

UC22.4: Modifica UID Satispay utente - Figura 3.5

Attori Principali: Utente.

Precondizioni: L'UID di Satispay dell'utente è salvato nel database con il vecchio valore.

Descrizione: L'utente modifica il proprio UID Satispay con il nuovo valore.

Postcondizioni: L'UID Satispay dell'utente è salvato nel database con il nuovo valore.

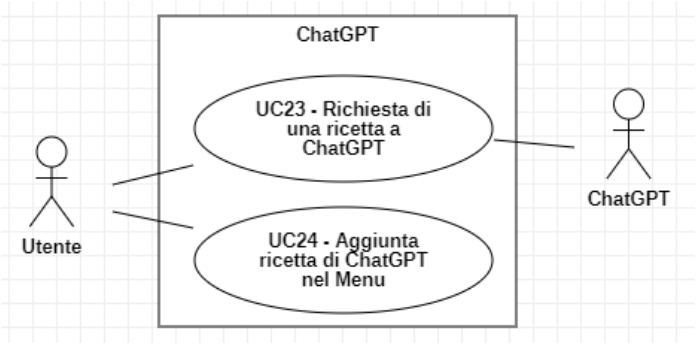

Figura 3.6: Use Case - ChatGPT

UC23: Richiesta di una ricetta a ChatGPT - Figura 3.6

Attori Principali: Utente, ChatGPT.

Precondizioni: L'utente vuole una nuova ricetta.

Descrizione: L'utente chiede a ChatGPT una ricetta.

Postcondizioni: ChatGPT restituisce una possibile ricetta all'utente.

UC24: Aggiunta ricetta di ChatGPT nel Menu - Figura 3.6

Attori Principali: Utente.

Precondizioni: ChatGPT ha consigliato una ricetta all'utente.

Descrizione: L'utente aggiunge la ricetta ricevuta da ChatGPT come nuovo piatto al menu e salva il piatto nel database.

Postcondizioni: La ricetta consigliata da ChatGPT è salvata nel database e visibile dal menu.

3.2 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.
Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli.
Il codice dei requisiti è così strutturato R(F/Q/V)(O/D/N) dove:

R = requisito

F = funzionale

Q = qualitativo

V = di vincolo

O = obbligatorio (necessario)

D = desiderabile

N = facoltativo

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali dall'1 al 7

Requisito	**Descrizione**	**Use Case**
RFO-1	L'utente effettua l'accesso all'app inserendo la propria password e la propria mail	UC1
RFO-2	L'utente si registra nel database inserendo il proprio nome, cognome, mail e password	UC2
RFO-3	Viene visualizzata la *cassa comune* salvata nel database nell'app	UC3
RFO-4	L'utente visualizza la propria *quota stornata* salvata nel database nell'app	UC4
RFO-5	Viene visualizzata la lista dei piatti proposti del giorno nell'app	UC5
RFO-6	Viene visualizzata la lista delle transazioni nell'app	UC6
RFO-7	L'utente aggiunge una nuova transazione nell'app, indicando i soldi e la data e salva la transazione nel database	UC7
RFO-8	L'utente amministratore aggiunge la spesa effettuata da uno stagista nell'app, indicando la data e quanto ha speso e lo salva nel database	UC7.1
RFO-9	L'utente indica la spesa che ha effettuato nell'app, riportando i soldi e la data e lo salva nel database	UC7.2
RFO-10	L'utente indica nell'app i soldi che ha inviato a un altro utente registrato nel database e salva la transazione nel database	UC7.3
RFO-11	L'utente elimina una transazione presente nel database dall'app	UC8
RFO-12	Viene visualizzato il menu che contiene la lista dei piatti dall'app	UC9

Tabella 3.2: Tabella del tracciamento dei requisiti funzionali dall'8 al 31

Requisito	Descrizione	Use Case
RFO-13	L'utente aggiunge un nuovo piatto nell'app, indicando il nome del piatto, gli ingredienti e la ricetta e lo salva nel database	UC10
RFO-14	L'utente elimina un piatto presente nel database dall'app	UC11
RFO-15	L'utente propone un piatto da mangiare a pranzo selezionandolo dal menu	UC12
RFO-16	L'amministratore visualizza la *quota pasto* dall'app	UC13
RFO-17	L'amministratore modifica la *quota pasto* dall'app e salva il nuovo valore nel database	UC14
RFO-18	L'amministratore visualizza la *quota stornata* degli stagisti dall'app	UC15
RFO-19	L'amministratore visualizza la lista con indicato i giorni di presenza degli stagisti dall'app	UC16
RFO-20	L'amministratore modifica la lista con indicato i giorni di presenza degli stagisti dall'app e salva le modifiche nel database	UC17
RFO-21	L'utente visualizza la lista con indicati i propri giorni di presenza a pranzo dall'app	UC18
RFO-22	L'utente modifica la lista con indicati i propri giorni di presenza a pranzo dall'app e salva le modifiche nel database	UC19
RFO-23	L'utente si disconnette dall'app	UC20
RFO-24	L'utente visualizza i propri dati dall'app	UC21
RFO-25	L'utente visualizza la propria mail dall'app	UC21.1
RFO-26	L'utente visualizza il proprio nome dall'app	UC21.2
RFO-27	L'utente visualizza il proprio cognome dall'app	UC21.3
RFO-28	L'utente visualizza il proprio UID Satispay dall'app	UC21.4
RFO-29	L'utente modifica i propri dati dall'app e salva le modifiche nel database	UC22
RFO-30	L'utente modifica la propria password dall'app e salva la nuova password nel database	UC22.1
RFO-31	L'utente modifica il proprio nome dall'app e salva il nuovo nome nel database	UC22.2
RFO-32	L'utente modifica il proprio cognome dall'app e salva il nuovo cognome nel database	UC22.3
RFO-33	L'utente modifica il proprio UID Satispay e salva il nuovo UID nel database	UC22.4
RFD-34	Viene chiesto a ChatGPT una possibile ricetta da proporre a pranzo	UC23
RFD-35	Si aggiunge la ricetta proposta da ChatGPT nel menu e si salva la ricetta nel database	UC24

Tabella 3.3: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Use Case
RQN-1	Il codice *front-end* deve essere coperto da test di unità	-

Tabella 3.4: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Use Case
RVO-1	L'applicazione deve essere sviluppata con il *framework* Flutter	-
RVO-2	L'applicazione deve essere sviluppata con la piattaforma Firebase	-
RVO-3	L'applicazione deve essere accessibile su cellulari con sistema operativo Android e iOS	-
RVO-4	La mail che l'utente deve utilizzare per registrarsi nel database e accedere all'app deve essere fornita da RiskAPP	-
RVO-5	La mail dell'utente non deve essere modificabile tramite app	-

Capitolo 4

Progettazione e codifica

Questo capitolo tratta della fase di progettazione e della fase di codifica dell'app, riportando tutto quello che è stato fatto e le difficoltà incontrate.

4.1 Progettazione

4.1.1 Struttura dell'app

Dopo una prima parte di stage, dove ho studiato le tecnologie riportate al secondo capitolo, ho pensato come sviluppare l'applicazione richiesta.
Di prassi, in RiskAPP si utilizza Figma per poter avere una idea più chiara del lavoro che si desidera fare, quindi per prima cosa ho progettato la grafica e la struttura dell'app con l'aiuto di questo software di progettazione.
Avendo una idea visiva, questo rendeva più facile spiegare al mio tutor come pensavo di impostare l'applicazione, andando poi a modificare e sistemare in base alle esigenze dell'azienda.

Nella Figura 4.1 ci sono tre schermate progettate in Figma, ovvero le schermate **Utente**, **Impostazioni** e infine **Home**.
Dai *mockup* capiamo che la struttura delle pagine è la seguente:

- una barra superiore, dove è possibile eseguire una o due azioni;
- una schermata con le informazioni interessate;
- una barra inferiore che permette di navigare tra le schermate, fatta eccezione per la schermata **Impostazioni** che non ha questa barra.

Tramite la barra inferiore è possibile navigare tra le schermate:

- **Home**, dove sarà visibile la *cassa comune*, la *quota stornata* dell'utente e infine il piatto del giorno (successivamente cambiato in *Proposte del giorno* per permettere la scelta di più piatti dal menu);
- **Spese**, che permette di visualizzare tutte le transazioni di tutti gli utenti e aggiungere delle nuove transazioni o eliminarle;

- **Menu**, dove è possibile consultare i piatti, aggiungerli oppure proporli come possibili piatti del giorno;
- **Utente**, la visualizzazione cambia tra utente semplice e utente amministratore. Quest'ultima è stata modificata durante la fase di codifica, ma principalmente serve per visualizzare e modificare le proprie presenze o, nel caso dell'amministratore, visualizzare e modificare le presenze degli stagisti o della *quota pasto*.

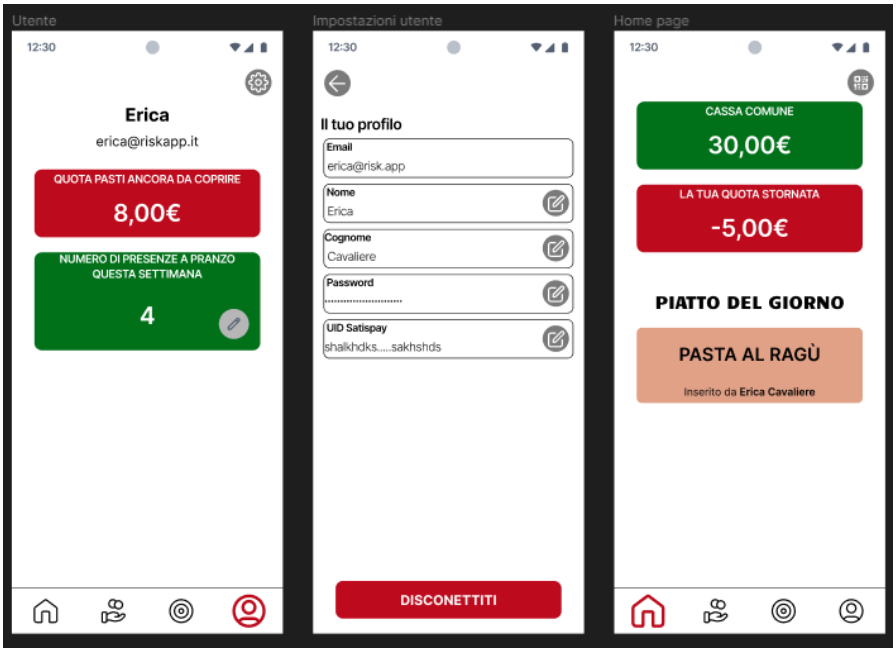

Figura 4.1: Alcune schermate progettate in Figma

La schermata **Impostazioni** è raggiungibile attraverso la schermata **Utente**, andando a toccare l'icona ad ingranaggio posta nella barra superiore.
Da **Impostazioni** è possibile modificare i dati dell'utente oppure permettere all'utente di disconnettersi dalla sessione corrente.

Sono state create diversamente anche le finestre **Accedi** (Figura 4.2) e **Registrati** (Figura 4.3).
In queste due schermate non sono presenti barre superiori o inferiori, ma solo una serie di campi da compilare e il pulsante verde Accedi o Registrati.
Accedi è la prima schermata che vede l'utente quando entra nell'app per la prima volta; per passare alla schermata **Registrati** bisogna toccare il link presente sotto al pulsante Accedi, dove è scritto il messaggio "*Sei nuovo? REGISTRATI*".
Per ritornare alla schermata **Accedi**, il procedimento è analogo, ovvero si tocca il link presente sotto il pulsante Registrati, dove è riportato il messaggio "*Hai già un account? ACCEDI*".
Invece, per andare in **Home** tramite entrambe le schermate appena descritte, bisognerà compilare correttamente i campi e poi toccare il pulsante verde presente.

In **Accedi** è presente il messaggio "*Hai dimenticato la password?*", questo dove-

va contenere un link che permetteva all'utente di recuperare la propria password, funzionalità prima prevista, poi ritenuta non più necessaria.

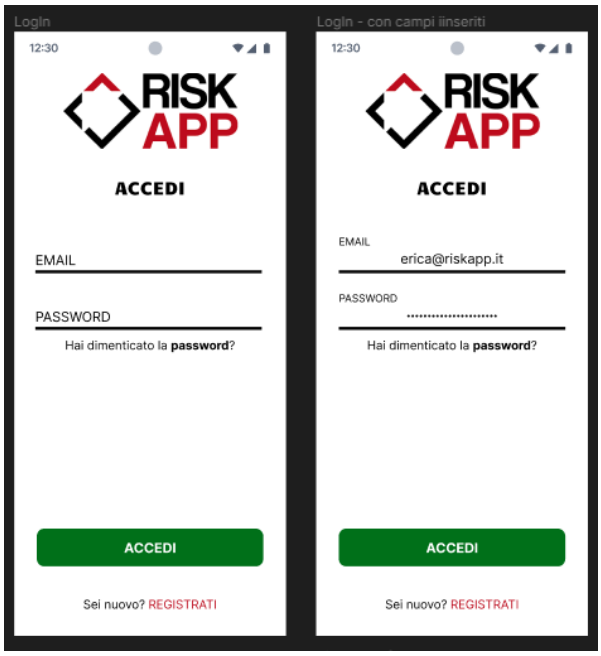

Figura 4.2: Schermata Accedi progettata in Figma

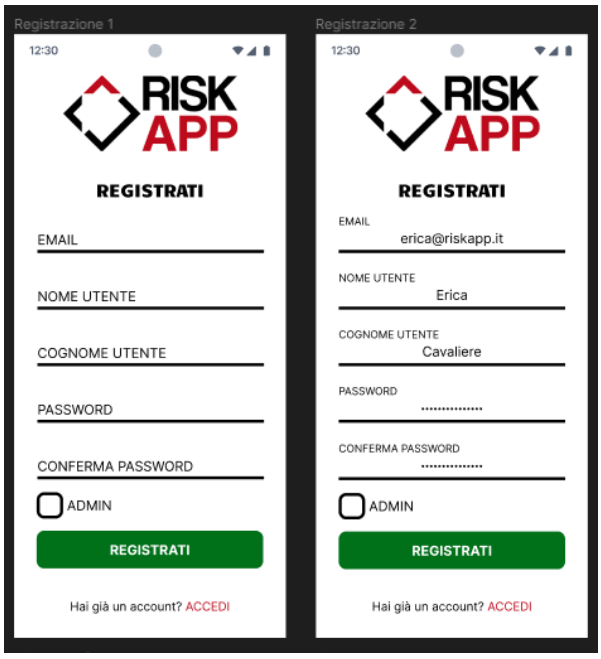

Figura 4.3: Schermata Registrati progettata in Figma

4.1.2 Database

Per quanto riguarda la base di dati (Figura 4.4), ho pensato a una struttura semplice, mettendo al centro l'utente che può inserire dei piatti, delle transazioni oppure segnare le proprie presenze.
Si è poi pensato ad un'entità isolata, che ha il solo scopo di contenere le variabili globali, ovvero la *cassa comune*, la *quota pasto* e la *quota stornata* degli stagisti.
Per quanto riguarda gli stagisti, inizialmente si pensava se considerarli come un utente oppure se rappresentarli come un'altra entità; alla fine, è stato deciso di lavorare in modo diverso, dato che gli stagisti si ipotizza che non utilizzino l'app.
Gli utenti amministratori possono aggiungere le spese e le presenze degli stagisti, mentre la loro *quota stornata* è stato deciso di indicarla nell'entità Cassa Comune, dato che si tratta di un dato unico per tutti gli stagisti.

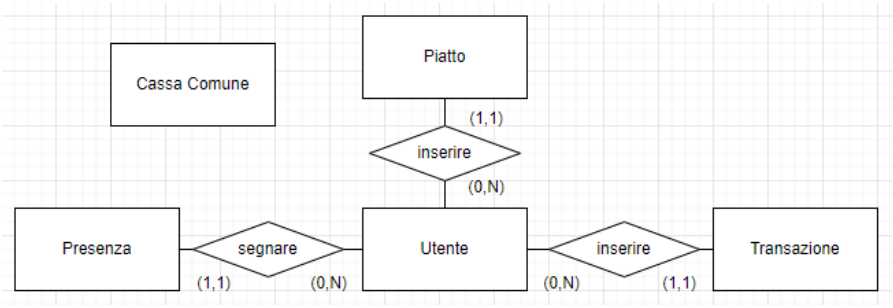

Figura 4.4: Il database progettato per l'applicazione

Lavorando con Firebase, un database NoSQL, i dati sono stati gestiti in modo leggermente diverso, ma rimanendo fedeli alle entità riportate in Figura 4.4.
Questo perchè Firebase organizza i dati in *raccolte*, ogni raccolta contiene dei *documenti* e ogni documento è composto da *campi*.
Se dovessimo tradurre a livello teorico:

- le *raccolte* sono le entità;
- i *documenti* sono le *tuple*, cioè se dovessimo rappresentare le entità come una tabella, un documento è una riga di informazioni dell'entità;
- i *campi* sono gli attributi, ovvero un singolo dato della *tupla*.

Con Firebase si possono gestire i dati in modo diverso da quanto descritto, perchè i *documenti* non sono rigidi, cioè i *documenti* all'interno di una *raccolta* possono avere quantità e tipi diversi di *campi*, permettendo una struttura dinamica e lasciando al programmatore la libertà di decidere come gestire le informazioni.

Per convenzione, è stato deciso di strutturare i dati come è stato descritto nell'elenco puntato, perchè risultava più semplice capire l'ordine delle informazioni e ha permesso una semplice gestione dei dati a livello di codice.
Di seguito si riporta la struttura finale del database, riportando per ogni **raccolta** i *campi* che ogni documento deve contenere.

cassaComune

- *cassaComune*, di tipo number (numerico); contiene il valore della *cassa comune*
- *quotaPasti*, di tipo number; contiene il valore attuale della *quota pasto*
- *quotaStornataStagisti*, di tipo number; contiene la *quota stornata* degli stagisti

Per questa raccolta esiste un solo *documento* con codice identificativo "cassaComune".

utenti

- *email*, di tipo string (stringa); riporta la mail dell'utente
- *nome*, di tipo string; riporta il nome dell'utente
- *cognome*, di tipo string; riporta il cognome dell'utente
- *UID*, di tipo string; questo rappresenta il codice UID Satispay dell'utente
- *quotaStornta*, di tipo string; indica la *quota stornata* dell'utente riferito
- *admin*, di tipo boolean (booleano); riporta se l'utente è un amministratore o un utente semplice

piatti

- *nome*, di tipo string; riporta il nome del piatto
- *ingredienti*, di tipo string; indica gli ingredienti necessari per preparare il piatto
- *ricetta*, di tipo string; riporta la ricetta del piatto riferito
- *propostoOggi*, di tipo timestamp; questo permette di capire quando è stata l'ultima volta che il piatto è stato proposto; se riporta la data odierna, il piatto dovrà comparire nella Home
- *utente*, di tipo string; serve per capire quale utente ha inserito il piatto

presenze

- *data*, di tipo timestamp; riporta la data di quando è stato presente l'utente
- *utente*, di tipo string; riporta la mail dell'utente a cui fa riferimento la presenza
- *quotaPasto*, di tipo number; indica quanto riportava la *quota pasto* il giorno in cui l'utente è stato presente a pranzo
- *stagisti*, di tipo boolean; se impostato a *true* vuol dire che la presenza indicata riporta la presenza degli stagisti e non dell'utente
- *numStagisti*, di tipo number; se *stagisti* è impostato a *true* riporta quanti stagisti erano presenti nella data indicata

transazioni

- *soldi*, di tipo number; riporta la quantità di soldi spesi
- *data*, di tipo timestamp; indica la data di quando è stata eseguita la transazione indicata
- *utente*, di tipo sting; riporta la mail dell'utente a cui fa riferimento la transazione
- *stagista*, di tipo boolean; indica se è stato lo stagista ad effettuare la spesa (*true*) o l'utente (*false*)
- *spesa*, di tipo boolean; se *true* indica che la transazione riportata è una spesa, altrimenti si tratta dell'invio dei soldi ad un altro utente
- *utenteRiceveInvioSoldi*, di tipo string; se *spesa* è impostato a *false*, questo riporta l'utente che riceve i soldi

4.2 Codifica

4.2.1 Struttura delle cartelle

Per il progetto è stato installato il *Software Development Kit (SDK)*[g] di Flutter nella versione 3.13.6.

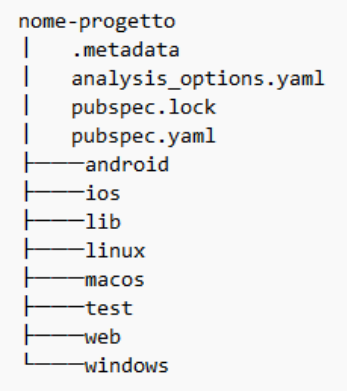

Figura 4.5: La struttura del progetto preimpostata da Flutter

La prima cosa che si nota quando si crea un progetto Flutter, è la struttura preimpostata dal *framework* (Figura 4.5):

- all'interno del file *.metadata* si trovano le proprietà del codice Flutter; questo file **non** bisogna modificarlo perchè contiene i metadati del progetto;
- il file *analysis_ options.yaml* serve per analizzare il codice Dart e controllare che non ci siano errori quando si compila il codice; come è intuibile, anche questo file **non** bisogna toccarlo;

- il file principale che controlla le librerie da installare è *pubspec.yaml*; se si desidera aggiungere un pacchetto, impostare un font specifico o anche indicare la cartella dove bisogna reperire i video e le immagini, bisogna indicarli dentro a questo file;
- per ogni piattaforma (Android, iOS, Linux, MacOS, Web e Windows), è presente una cartella apposita con all'interno tutto l'occorrente per far funzionare il progetto nel sistema operativo desiderato;
- il codice principale lo si scrive all'interno della cartella *lib*;
- Flutter preimposta la cartella *test* dove poter svolgere i test di unità; purtroppo non sono stati eseguiti test durante lo stage, perchè non c'è stato abbastanza tempo per testate il codice prodotto.

Per questo progetto, ho lavorato principalmente nella cartella *lib* e ho creato una cartella *assets* per inserire il logo dell'azienda RiskAPP, visibile nella pagine Accedi e Registrati dell'app; se fossero state presenti altre immagini, sarebbero state inserite all'interno di questa cartella.
Poche volte ho toccato le cartelle *android* e *ios*, principalmente per sistemare qualche libreria di Flutter che dava problemi su uno dei due sistemi operativi.

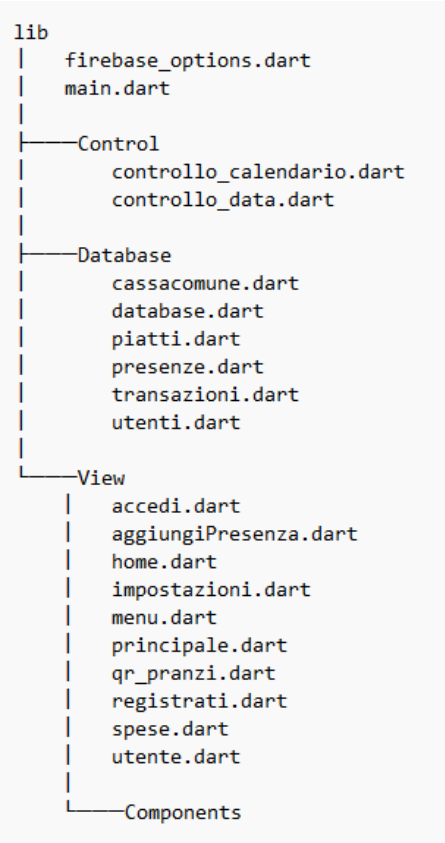

Figura 4.6: La struttura della cartella lib

Come si può vedere in Figura 4.6, ho creato tre cartelle per suddividere i file:

- nella cartella **Control**, sono state definite le classi per poter gestire le variabili locali dell'app, che sono solo di supporto per poter gestire alcune informazioni, come per esempio impostare la data nel formato italiano, implementato nella classe ControlloData() creata nel file *controllo_ data.dart*;
- all'interno di **Database**, sono presenti tutte le classi con le funzioni appropriate per poter comunicare e gestire il database di Firebase;
- all'interno della cartella **View**, viene gestita la parte grafica dell'app; per ogni pagina è stata creato un file apposito, l'unica eccezione è per *principale.dart* che ha il solo scopo di visualizzare la barra inferiore dell'app.

Il file *firebase_ options.dart* contiene le istruzioni fondamentali per collegare il progetto alla console di Firebase e viene generato automaticamente da *FlutterFire CLI*[g], un tool che mette a disposizione diversi comandi per installare *FlutterFire* e permettere di collegare il proprio progetto a Firebase.
In *main.dart* si decide quale pagina deve visualizzare l'utente quando apre l'app, in base se è stato eseguito l'accesso le volte precedenti o no.

Figura 4.7: La struttura della cartella Components

In **View** è presente la cartella **Components** (Figura 4.7), questa contiene le componenti, ovvero degli elementi preimpostati che vengono utilizzati più volte all'interno del codice.
Per esempio, il file *caricamento.dart* definisce la schermata che dovrà essere visibile mentre viene eseguito un caricamento dei dati da Firebase; questo componente viene richiamato da quasi tutte le pagine create in **View**.

Una parte delle componenti create gestiscono il testo e i relativi font, per l'esattezza si tratta dei file a partire da *sottotitolo.dart* fino all'ultimo presente in elenco, cioè *titolo_ badge.dart*.
Anche il file *caricamento_ messaggi.dart* definisce il testo altrnativo che dovrà essere visibile mentre vengono caricati solo alcune informazioni dal database.

Sono stati creati diversi componenti per permettere di visualizzare i dati, si tratta di tutti i file nell'elenco della cartella **Components** che iniziano con la parola *badge*. Questo perchè, per ogni schermata dell'app, i dati compariranno in modo leggeremente diverso oppure è possibile eseguire delle azioni che in altre schermate non sono presenti. Per esempio, i file *badge_ piatto.dart* e *badge_ piatto_ modifica.dart* riportano come devono apparire i piatti nel Menu, distinguendo tra i piatti inseriti dall'utente, quindi modificabili, e quelli creati da altri utenti, quindi i piatti che si possono solo visualizzare. In entrambi i casi, dovrà comparire il pulsante PROPONI, che permette di proporre un piatto per il pranzo odierno e, se cliccato, farà visualizzare per tutti gli utenti il piatto nella lista dei piatti proposti.

Anche per la pagina Impostazioni sono stati creati dei componenti distinti (*box_ impostazioni.dart* e *box_ impostazioni_ modale.dart*), mentre i seguenti file rappresentano dei componenti unici, modificabili in base all'uso:

- *calendario.dart* definisce il calendario utilizzato per gestire le presenze dell'utente o dei stagisti;
- *chiamata_ modale.dart* permette di aprire una finestra che si posiziona sopra la schermata dove l'utente l'ha richiamata, non coprendola totalmente; in questo componente bisogna definire il messaggio o i dati da visualizzare e si definisce anche l'icona del pulsante che permette di visualizzare questa finestra; è stata utilizzata per permettere di aggiungere, eliminare o modificare alcuni dati tramite app;
- *pulsante_ principale.dart* consente di visualizzare un bottone, bisognerà definire il testo, il colore e l'azione che deve svolgere il bottone se viene cliccato;
- *barra_ filtro.dart* definisce la barra superiore che deve essere visualizzata nelle schermate Spese e Menu; a destra della barra riporta un pulsante che consente di selezionare il filtro da applicare all'elenco presente in schermata, mentre il bottone a sinistra è modificabile dal programmatore.

4.2.2 Le librerie di Firebase

Quando si collega un progetto a Firebase, si scaricano i pacchetti *Firebase CLI* e *FlutterFire CLI*, questi permettono di collegare il progetto alla console di Firebase e creano il file *firebase_ option.dart* che dovrà poi essere importato nel file *main.dart*. Infine, basta scrivere le righe di codice riporte in Figura 4.8 per poter permettere ad ogni piattaforma di interagire con il database.

```
void main() async {
  WidgetsBinding widgetsBinding = WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
```

Figura 4.8: Istruzioni per collegare Firebase

Quando si eseguono i passaggi appena descritti, si installa in automatico il pacchetto *firebase_ core*, ma per poter permettere l'autenticazione degli utenti e lavorare con il database, sono stati installati manualmente i pacchetti *firebase_ auth* e *cloud_ firestore*.

Ci sono diversi modi che Firebase Authentication mette a disposizione per la registrazione di un utente, ma per questo progetto è stato adottato il metodo di autenticazione tramite mail e password.
Per permettere la registrazione, l'accesso e la disconnessione di un utente, è stato creata la classe *Database* che al suo interno contiene solo i metodi con le funzioni appena descritte (in Figura 4.9 sono riportate le funzioni *accedi* e *disconettiti*).

```
static Future<void> accedi (String email, String password) async{
  try {
    UserCredential userCredential = await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: email,
        password: password
    );
    _err = "";
    print("Utente loggato");
  } on FirebaseAuthException catch (e) {
    if (e.code == 'invalid-email'){
      print('The email is invalid');
    } else if (e.code == 'INVALID_LOGIN_CREDENTIALS'){
      print('INVALID_LOGIN_CREDENTIALS');
    }
    _err = e.code;
  }
}

static Future<void> disconettiti() async {
  await FirebaseAuth.instance.signOut();
  print("Utente disconesso");
}
```

Figura 4.9: Le funzioni di accesso e di disconnessione di un utente

Per poter lavorare con il Cloud di Firestore, è stata creata una classe per ogni entità (CassaComune, Utenti, Piatti, Presenze e Transazioni).
Ogni classe contiene un metodo per creare una nuova istanza (Figura 4.10), metodi *set* e *get* per ogni campo (Figura 4.11) e alcune funzioni di supporto.

```
static CollectionReference utenti = FirebaseFirestore.instance.collection('utenti');

static Future<void> aggiungiUtente(String email, String nome, String cognome, String UID, bool admin) {
  // Call the user's CollectionReference to add a new user
  return utenti
      .add({
    'email': email,
    'nome': nome,
    'cognome': cognome,
    'UID': UID,
    'admin': admin,
    'quotaStornata': 0,
  })
      .then((value) => print("User Added"))
      .catchError((error) => print("Failed to add user: $error"));
}
```

Figura 4.10: La funzione di aggiunta di un utente nel database

```
static Future<String> getUIDSatispayUtente (String idUtente) async {
  String uid = await utenti.doc(idUtente).get().then((DocumentSnapshot documentSnapshot) {
    if (documentSnapshot.exists) {
      return documentSnapshot['UID'].toString();
    } else {
      return '';
    }
  });
  return uid;
}

static Future<void> setNomeUtente (String idUtente, String nome) async {
  await utenti.doc(idUtente).update({'nome':nome});
}
```

Figura 4.11: Una funzione *get* e *set* della classe *Utenti*

Attraverso la console di Firebase è possibile vedere gli utenti che si sono registrati, i dati presenti nel database ed è possibile anche modificare i permessi di lettura o modifica dei dati, per un controllo più attento anche a livello di codice.

4.2.3 Il calendario delle presenze

Per permettere la gestione delle presenze, è stato installato il pacchetto *table_ calendar*, che permette di creare il *widget* TableCalendar, cioè l'elemento grafico che vediamo in Figura 4.12.

Figura 4.12: Il calendario che l'utente utilizza per gestire le presenze

Il *widget* offre una serie di opzioni che consentono di modificarlo, come per esempio scegliere se visualizzare il calendario nel formato settimanale (come in immagine), con due settimane oppure tutto un mese.
Si può modificare come viene visualizzata la data selezionata (anche la data odierna) attraverso l'uso del *CalendarBuilder*: si tratta di un costruttore (*builder*) con il compito di impostare la grafica dell'elemento indicato; in questo caso ha il compito di impostare la grafica delle date evidenziate.

Questo calendario è presente nella pagina Utente, dove sono visibili i tre *widget* grigi, il primo apre il calendario per modificare le proprie presenze (Figura 4.14), il secondo, visibile per gli amministratori, apre il calendario per modificare e monitorare le presenze degli stagisti.
Il terzo *widget* consente solo di visualizzare e modificare la *quota pasto*, anche questo visbile solo agli amministratori.
Non sono stati inseriti altri calendari nell'applicazione.

Una funzione fondamentale di TableCalendar è *onDaySelected* (Figura 4.13): questa permette di modificare il calendario selezionato, rendendo la selezione iterativa; se non viene definita questa funzione, il calendario rimane statico e non sarà possibile cambiare la data selezionata, ma rimarrà evidenziata la data indicata inizialmente nella variabile *focusedDay*.

```
onDaySelected: (DateTime day, DateTime focusDay){
  setState(() {
    today = day;
    ControlloCalendario.setDay(day);
    onDaySelected();
  });
},
```

Figura 4.13: La funzione onDeySelected definita per il progetto

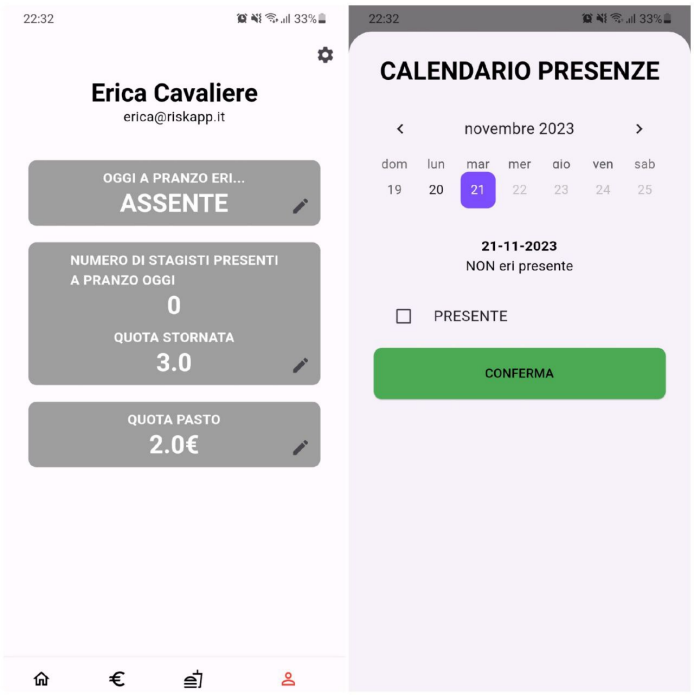

Figura 4.14: La schermata Utente e il calendario delle presenze

Figura 4.15: Le schermate Spese e Menu

4.2.4 Le schermate Spese e Menu

Per questo progetto è stata richiesta un'app che permettesse, oltre un controllo e una gestione delle presenze, anche un controllo delle spese e la presenza di un menu comune per tutti, dove è possibile proporre un piatto per il pranzo odierno (Figura 4.15).
Le due schermate che si occupano di queste funzioni, sono la schermata Spese, che permette di monitorare, aggiungere o eliminare le transazioni, e la schermata Menu, che permette di selezionare, aggiungere o proporre un piatto.

La struttura di queste due pagine è uguale alla struttura delle pagine Home e Utente, ovvero contengono una barra superiore, una barra inferiore per navigare e una schermata, ma se si analizza le due pagine Spese e Menu, si nota che sono state progettate in modo molto simile tra loro.
La barra di navigazione non cambia, per questo ci concentreremo di più sulla barra superiore e alla schermata principale.

- Tramite la barra superiore è possibile aggiungere un elemento alla lista oppure utilizzare il filtro per visualizzare solo alcuni elementi specifici.
- Nella schermata principale è possibile monitorare una lista di dati, nel caso di Spese si tratta della lista transazioni, per il Menu invece si tratta di tutti i piatti salvati dagli utenti, sempre gestita come una lista.

In Spese, le transazioni sono di tre colori distinti, ovvero verde nel caso in cui l'utente riceve dei soldi da altri utenti, rosso per le spese o se l'utente ha inviato i soldi ad altri e, infine, grigio per tutti gli altri casi.
Le transazioni sono inizialmente in ordine di data decrescente, ma è possibile cambiare l'ordine e i dati visualizzati tramite il filtro.
Le transazioni inserite dall'utente portano l'icona di un cestino, se cliccato permette di aprire una chimata modale di conferma di eliminazione della transazione; per tutti gli altri *badge* che non sono stati inseriti dall'Utente, questa icona non compare.
In ogni badge è visibile chi effettua la transazione, la data di esecuzione, i soldi spesi (si tratta del font più grande) e infine se l'utente ha effettuato una spesa o ha inviato i soldi a qualcuno.

Nel Menu, i piatti si presentano tutti uguali, ovvero sono dei *badge* a sfondo blu con un pulsante verde con scritto PROPONI.
Tutti i piatti inizialmente sono in ordine alfabetico, ma è possibile cambiare l'ordine o guardare dei piatti specificando tramite il filtro.
Se viene premuto il pulsante PROPONI, il piatto viene aggiunto alla lista dei piatti proposti visibile in Home, ma il piatto non scomparirà dal menu; l'utente quando clicca il pulsante, verrà riportato direttamente in Home per visualizzare la lista aggiornata.
Ogni badge riporta il nome del piatto e l'utente che lo ha inserito; se si preme lo sfondo blu, verrà aperta una chiamata modale che riporta gli ingredienti, la ricetta e eventualmente un'area per modificare i dati del piatto.

Quando si effettua una aggiunta o si applica un filtro, il lavoro che l'app svolge è aggiornare il database o richiedere degli elementi specifici nel cloud di Firebase, per poi aggiornare graficamente la lista interessata.
Ogni modifica in Firebase viene ripotata in tempo reale su tutti i dispositivi che utilizzano l'app; per permettere questo, sono stati utilizzati gli **StreamBuilder**.
Dato uno *stream* di dati, lo **StreamBuilder** si occupa di creare e aggiornare dei dati

specifici; in questo caso, per il progetto creato, si occupa di aggiornare le liste presenti nelle schermate Spese e Menu.

Lo **StreamBuilder** è stato utilizzato anche nella Home, per permettere all'utente di monitorare in tempo reale i piatti proposti, la *cassa comune* e la propria *quota stornata*, quest'ultimi due sono in continuo aggiornamento in base alle modifiche riportate nella schermata Spese o tramite le presenze segnate nel calendario (per ogni presenza viene effettuata una modifica pari alla quantità di soldi indicata nella *quota pasto*).
È stato utilizzato lo **StreamBuilder** anche nella schermata Utente per il controllo in tempo reale della *quota stornata* degli stagisti.

4.2.5 Modificare il nome e il logo

Quando si crea un'applicazione, di default Flutter imposta il suo logo.
Per poter modificare e utilizzare il proprio logo è molto semplice, prima di tutto bisogna creare più formati dell'immagine che si desidera attribuire all'app, poi bisogna riportarli tutti all'interno della cartella del sistema operativo interessato e preimpostato da Flutter.

- Per Android bisogna salvare il logo e tutti i suoi formati nel percorso *android/app/src/main/res*
- Per iOS bisogna salvare il logo e tutti i suoi formati nel percorso *ios/Runner/Assets.xcassets/AppIcon.appiconset*

Il logo per questo progetto è stato creato in Figma (Figura 4.16) e tutti i suoi formati sono stati creati utilizzando il sito App Icon Generator.

Il procedimento per modificare il nome dell'applicazione non si discosta molto dal procedimento appena descritto.

- Per Android bisogna modificare il parametro *label* presente nel file *AndroidManifest.xml* che si trova nel percorso *android/app/src/main*.
- Per iOS bisogna modificare il parametro *CFBundleDisplayName* nel file *Info.plist* presente nel percorso *ios/Runner*.

Figura 4.16: Il nome e il logo dell'applicazione creata

4.2.6 Aprire l'app tramite QrCode

Per poter gestire le presenze in modo veloce, si pensava di utilizzare un QrCode che permettesse di segnare la presenza dell'utente quando veniva scannerizzato.
Purtroppo, non è stato possibile implementare questa funzione perchè sono state incontrate diverse difficoltà nel creare quanto richiesto, ma è stato un caso di studio interessante che potrebbe essere approfondito in futuro.

Per poter creare un qualsiasi QrCode, è stato installato il pacchetto *qr_flutter*; quest'ultimo permette di creare il *widget QrImageView*, che si occupa di far vedere direttamente nell'applicazione un QrCode, generato da una stringa impostata dal programmatore.
Il dubbio a questo punto sorge spontaneo: se bisogna passare una parola, una frase, un qualsiasi testo per creare un QrCode, qual è la stringa che permette di aprire la propria applicazione Flutter?
La risposta, in realtà, è molto semplice a parole, ma nella pratica incontra qualche difficoltà.
Per aprire un'applicazione, bisogna creare un URL identificativo per l'app e questa sarà la stringa che bisognerà passare a *QrImageView* per generare il QrCode desiderato.

Creare un URL è molto semplice, ci sono diversi servizi online che permettono di creare il proprio sito e ottenere così un indirizzo web.
In accordo con il mio tutor, è stato scelto di utilizzare il servizio Hosting di Firebase; il passaggio successivo consiste nel collegare il proprio progetto Flutter all'URL creato.

Per configurare correttamente la propria app Flutter, bisogna utilizzare la navigazione tramite *route*, ovvero associando un percorso per ogni pagina che si desidera aprire, oppure installando la libreria *go_route*.
Questo è stato il mio primo errore per testare il QrCode, perchè per gestire la navigazione tra pagine, ho utilizzato il *widget Navigator*; per questo motivo è stata installata successivamente la libreria richiesta e ho creato un file di test, ovvero una schermata contenente un messaggio di conferma presenza.

Dopo aver configurato il percorso di ogni schermata, bisogna modificare il file *AndroidManifest.xml* per Android e il file *Info.plist* per iOS, aggiungendo delle istruzioni specifiche che permettono la navigazione tramite i **Deep Link**.

I **Deep Link** sono un modo semplice per connettere le pagine di un'app.
Con questo metodo è possibile collegare la propria app ad altre, ma anche direzionare gli utenti a specifiche pagine in-app.
Questa tecnologia aiuta l'ecosistema mobile a diventare più connesso, piuttosto che creare una serie di applicazioni che non possono "comunicare" tra di loro.
Da non confondere con i **Dynamic link**, che sono sempre dei **Deep Link**, ma permettono di personalizzare il comportamento di questi.
Ad esempio, lo stesso link può portare ad una determinata sezione dell'app per Android ma ad una diversa sezione dell'app per iOS oppure, se l'utente non ha l'app installata, può spedirlo allo store (di Google, o di Apple) per scaricare l'applicazione.
Dopo l'installazione il link porta a termine il suo compito aprendo la sezione giusta dell'app oppure, se preferiamo, possiamo chiedere al link di inviare al nostro sito Web l'utente che non ha l'app installata sul suo smartphone.

Dopo aver configurato la propria applicazione, creato l'URL e connesso l'app al sito, ora è possibile creare un QrCode che permette di aprire direttamente l'applicazione o, se l'app non è installata, aprire la pagina web.
Questo è tutto quello che sono riuscita a fare, ho seguito tutti i passaggi appena descritti, creato il QrCode passando come stringa l'URL dell'app e, se si scannerizza il QrCode con un qualsiasi lettore di codici Qr, aprire l'applicazione.

Considerando quanto fatto, possiamo definirlo un successo, perchè sono riuscita a studiare e a fare una cosa per me nuova, ma non una vittoria, perchè il QrCode non doveva solo aprire l'applicazione, doveva anche direzionare l'utente ad una pagina specifica e permettere così di segnare la propria presenza a pranzo nella giornata odierna.
Per questo ultimo passaggio sono state incontrate diverse difficoltà, tanto che abbiamo deciso di abbandonare l'idea, dato che è già possibile segnare manualmente la propria presenza tramite il calendario, ma sarà una sfida che affronterò ancora in futuro.

Capitolo 5

Conclusioni

In conclusione a tutto il percorso di stage, in questo capitolo viene fatto un resoconto di tutti gli obbiettivi raggiunti, riportando anche una valutazione personale di tutto il lavoro svolto, analizzando difficoltà successi e possibili punti da migliorare in futuro nel progetto.

5.1 Valutazione personale

Il progetto che ho sviluppato tratta di un prototipo di una applicazione per gestire i pranzi aziendali.
Ho lavorato in solitaria, ma con il sostegno del mio tutor che è sempre stato disponibile quando chiedevo aiuto o consigli.

Durante la mia permanenza in azienda, ci sono stati altri due stagisti che hanno lavorato al mio stesso progetto, ma concentrandosi su funzioni diverse.
Nel mio caso, dovevo creare una gestione automatizzata per permettere il controllo della presenza a pranzo, un secondo stagista ha lavorato con ChatGPT per poter creare e eventualmente aggiungere dei piatti nel menu, mentre l'ultimo ha integrato Satispay all'app, per poter così gestire meglio le transazioni.

Il motivo di questa distinzione è per creare delle piccole bozze di una applicazione importante, capendo se le funzioni sviluppate possono essere utili per i dipendenti dell'azienda o se sono da scartare e quindi da non implementare nell'applicazione finale.

Nonostante si tratta di una bozza, sono lo stesso contenta di aver affortontato questo progetto, perchè mi ha permesso di conoscere il linguaggio Dart con Flutter e il funzionamento di Firebase.
Le prime due settimane sono state spese sullo studio delle tecnologie, poi ho dedicato un paio di settimene alla progettazione su Figma e poi ho sempre lavorato sulla codifica dell'app.

Essendo un linguaggio nuovo per me, durante la fase di codifica ho avuto un po' di difficoltà, in particolare con Firebase, perchè non riuscivo a capire come collegare la console Firestore e lavorare sul database tramite codice Dart; ho risolto dopo una settimana di ricerche e di prove, anche con l'aiuto del mio tutor aziendale.

Valutando tutto il lavoro svolto, ci sono alcune cose dell'app che cambierei, un esempio è la pagina transazione, andrei a gestire in modo diverso la visualizzazione dei dati nella lista, togliendo il messaggio di spesa o di invio soldi.
Anche il calendario modificherei, nonostante risponda alle esigenze dell'utente, se viene utilizzato tutti i giorni e se in un futuro si decide di fare delle modifche in uno o più giornate passate, questo può risultare pesante perchè, per come ho progettato il lavoro, si può modificare solo una data alla volta, facendo risultare all'utente un lavoro noioso e macchinoso.
È stata una sfida anche implementare il QrCode, anche se non sono riuscita a implementare la funzione desiderata, sono lo stesso contenta del risultato ottenuto.

Nonostante ci siano ancora molti punti su cui lavorare, sono riuscita a rispettare gli obiettivi obbligatori, sviluppando le funzioni principali richieste.

5.2 Raggiungimento degli obiettivi

In riferimento alle notazioni indicate nel primo capitolo e nel terzo capitolo e in riferimento alle Tabelle 1.1, 3.1, 3.2, 3.3 e 3.4, di seguito sono riportati gli obbiettivi e i requisiti con il loro stato di completamento.
Alcuni obbiettivi e alcuni requisiti non sono stati soddisfatti perchè è stato deciso di impiegare più tempo per il raggiungimento degli obiettivi obbligatori, dedicando più tempo allo studio e cercando di migliorare alcune funzioni dell'applicazione richiesta.

Tabella 5.1: Tabella degli obiettivi raggiunti

Obiettivo	Descrizione	Stato
O01	Accesso tramite credenziali	Soddisfatto
O02	Pannello di controllo degli utenti	Soddisfatto
O03	Modifica o aggiunta delle spese	Soddisfatto
O04	Monitoraggio della cassa comune	Soddisfatto
O05	Controllo presenza in azienda di una persona durante i pranzi	Soddisfatto
O06	Impostare il menu del giorno	Soddisfatto
O07	Scelta di un piatto dal menu	Soddisfatto
D01	Integrazione di ChatGPT per consigliare alcune ricette	Non soddisfatto
F01	Test a livello di frontend	Non soddisfatto

Tabella 5.2: Tabella del tracciamento dei requisiti funzionali dall'1 al 17 raggiunti

Requisito	**Descrizione**	**Stato**
RFO-1	L'utente effettua l'accesso all'app inserendo la propria password e la propria mail	Soddisfatto
RFO-2	L'utente si registra nel database inserendo il proprio nome, cognome, mail e password	Soddisfatto
RFO-3	Si visualizza la *cassa comune* salvata nel database nell'app	Soddisfatto
RFO-4	L'utente visualizza la propria *quota stornata* salvata nel database nell'app	Soddisfatto
RFO-5	Si visualizza la lista dei piatti proposti del giorno nell'app	Soddisfatto
RFO-6	Si visualizza la lista delle transazioni nell'app	Soddisfatto
RFO-7	L'utente aggiunge una nuova transazione nell'app, indicando i soldi e la data e salva la transazione nel database	Soddisfatto
RFO-8	L'utente amministratore aggiunge la spesa effettuata da uno stagista nell'app, indicando la data e quanto ha speso e lo salva nel database	Soddisfatto
RFO-9	L'utente indica la spesa che ha effettuato nell'app, riportando i soldi e la data e lo salva nel database	Soddisfatto
RFO-10	L'utente indica nell'app i soldi che ha inviato a un altro utente registrato nel database e salva la transazione nel database	Soddisfatto
RFO-11	L'utente elimina una transazione presente nel database dall'app	Soddisfatto
RFO-12	Si visualizza il menu che contiene la lista dei piatti dall'app	Soddisfatto
RFO-13	L'utente aggiunge un nuovo piatto nell'app, indicando il nome del piatto, gli ingredienti e la ricetta e lo salva nel database	Soddisfatto
RFO-14	L'utente elimina un piatto presente nel database dall'app	Soddisfatto
RFO-15	L'utente propone un piatto da mangiare a pranzo selezionandolo dal menu	Soddisfatto
RFO-16	L'amministratore visualizza la *quota pasto* dall'app	Soddisfatto
RFO-17	L'amministratore modifica la *quota pasto* dall'app e salva il nuovo valore nel database	Soddisfatto

Tabella 5.3: Tabella del tracciamento dei requisiti funzionali dal 18 al 35 raggiunti

Requisito	Descrizione	Stato
RFO-18	L'amministratore visualizza la *quota stornata* degli stagisti dall'app	Soddisfatto
RFO-19	L'amministratore visualizza la lista con indicato i giorni di presenza degli stagisti dall'app	Soddisfatto
RFO-20	L'amministratore modifica la lista con indicato i giorni di presenza degli stagisti dall'app e salva le modifiche nel database	Soddisfatto
RFO-21	L'utente visualizza la lista con indicati i propri giorni di presenza a pranzo dall'app	Soddisfatto
RFO-22	L'utente modifica la lista con indicati i propri giorni di presenza a pranzo dall'app e salva le modifiche nel database	Soddisfatto
RFO-23	L'utente si disconnette dall'app	Soddisfatto
RFO-24	L'utente visualizza i propri dati dall'app	Soddisfatto
RFO-25	L'utente visualizza la propria mail dall'app	Soddisfatto
RFO-26	L'utente visualizza il proprio nome dall'app	Soddisfatto
RFO-27	L'utente visualizza il proprio cognome dall'app	Soddisfatto
RFO-28	L'utente visualizza il proprio UID Satispay dall'app	Soddisfatto
RFO-29	L'utente modifica i propri dati dall'app e salva le modifiche nel database	Soddisfatto
RFO-30	L'utente modifica la propria password dall'app e salva la nuova password nel database	Soddisfatto
RFO-31	L'utente modifica il proprio nome dall'app e salva il nuovo nome nel database	Soddisfatto
RFO-32	L'utente modifica il proprio cognome dall'app e salva il nuovo cognome nel database	Soddisfatto
RFO-33	L'utente modifica il proprio UID Satispay e salva il nuovo UID nel database	Soddisfatto
RFD-34	Viene chiesto a ChatGPT una possibile ricetta da proporre a pranzo	Non soddisfatto
RFD-35	Si aggiunge la ricetta proposta da ChatGPT nel menu e si salva la ricetta nel database	Non soddisfatto

Tabella 5.4: Tabella del tracciamento dei requisiti qualitativi raggiunti

Requisito	Descrizione	Stato
RQN-1	Il codice *front-end* deve essere coperto da test di unità	Non soddisfatto

Tabella 5.5: Tabella del tracciamento dei requisiti di vincolo raggiunti

Requisito	Descrizione	Stato
RVO-1	L'applicazione deve essere sviluppato con il *framework* Flutter	Soddisfatto
RVO-2	L'applicazione deve essere sviluppato con la piattaforma Firebase	Soddisfatto
RVO-3	L'applicazione deve essere accessibile su cellulari con sistema operativo Android e iOS	Soddisfatto
RVO-4	La mail che l'utente deve utilizzare per registrarsi nel database e accedere all'app deve essere fornita da RiskAPP	Soddisfatto
RVO-5	La mail dell'utente non deve essere modificabile tramite app	Soddisfatto

Acronimi e abbreviazioni

CLI Command Line Interface. 47

IDE Integrated Development Environment. 7, 47

SDK Software Development Kit. 29, 47

UI User Interface. 4, 48

UML Unified Modeling Language. 10, 48

UX User Experience. 4, 48

WCAG Web Content Accessibility Guidelines. 5, 48

Glossario

Build indica la trasformazione del codice in un prodotto software eseguibile. 7, 8

Cassa Comune viene utilizzato questo termine per indicare i fondi dati dagli operatori aziendali per coprire i pasti. 2, 12, 21, 24, 27, 28, 38, 43

CLI interfaccia a riga di comando. 31, 33, 46

Componenti sono un insieme di *widget* e di elementi che insieme costituiscono un prodotto software. 5

Dart linguaggio di programmazione *open-source* sviluppato da Google. È il linguaggio principale utilizzato per scrivere applicazioni con *Flutter*. Dart è noto per la sua velocità ed efficienza nella creazione di applicazioni mobili e web. Risulta inoltre staticamente tipizzato, cioè consente una dichiarazione esplicita dei tipi delle variabili e garantisce maggiore robustezza in programmazione. 2, 47

Firebase piattaforma di sviluppo di app mobile di Google che offre una serie di servizi tra cui *database* in tempo reale, autenticazione utente, *hosting* di applicazioni e molto altro. È ampiamente utilizzato per la costruzione di app mobile e web in modo rapido e scalabile, grazie alle funzionalità *cloud*, di notifica e di monitoraggio in *real time*. 2

Flutter *framework open-source* di Google per lo sviluppo di applicazioni mobile, desktop e webapp utilizzando il linguaggio *Dart*. È basato su *widget* personalizzabili, puntando su un rapido sviluppo, eccellenti performance, una comunità attiva e supporto per molte piattaforme. 2, 4, 5, 47

IDE è un ambiente di sviluppo integrato che supporta i programmatori nello sviluppo e nel *debug* del codice. 7, 8, 46

Quota Pasto indica il quantitativo di soldi che ogni utente deve dare per ogni pranzo effettuato in azienda. 15, 22, 25, 27, 28, 35, 38, 43

Quota Stornata indica i soldi che il singolo utente deve dare o ricevere dagli altri utenti per i pasti effettuati e le spese sostenute. 2, 12, 15, 21, 22, 24, 27, 28, 38, 43, 44

SDK è un insieme di strumenti che consente lo sviluppo di software o firmware per una specifica piattaforma. 46

UI indica l'interfaccia grafica che viene utilizzata per le comunicazioni tra uomo e macchina. 7, 46

UML in ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 46

UX indica l'insieme di sensazioni e ricordi che una persona prova quando si rapporta con un prodotto, cioè tutti gli aspetti che condizionano il prodotto per consentire all'utente di utilizzarlo e capirlo con facilità. 46

WCAG si tratta di una serie di linee guida per l'accessibilità, fornisce una serie di criteri tecnici per rendere siti web, applicazioni e altri contenuti facilmente utilizzabili da tutti i tipi di utente. 46

Bibliografia

Riferimenti bibliografici

Ken Schwaber, Jeff Sutherland. *La Guida Scrum - La Guida Definitiva a Scrum: Le Regole del Gioco*. Novembre 2020.

Siti web consultati

Change Flutter App Launcher Icon. URL: https://medium.com/flutter-community/change-flutter-app-launcher-icon-59c31bcd7554.

Cloud Firestore. URL: https://firebase.flutter.dev/docs/firestore/usage/.

Display Dynamic Events At Calendar In Flutter. URL: https://medium.flutterdevs.com/display-dynamic-events-at-calendar-in-flutter-22b69b29daf6.

Everything about the BottomNavigationbar in Flutter. URL: https://medium.com/flutter-community/everything-about-the-bottomnavigationbar-in-flutter-e99e5470dddb.

Figma Learn. URL: https://help.figma.com/hc/en-us.

Figma Tutorial. URL: https://help.figma.com/hc/en-us/sections/4405269443991-Figma-for-Beginners-tutorial-4-parts-.

Firebase Autentication. URL: https://firebase.flutter.dev/docs/auth/usage/.

Firebase Collegamenti dinamici. URL: https://firebase.google.com/docs/dynamic-links?hl=it.

Firebase collegamenti dinamici in un'app Flutter. URL: https://firebase.google.com/docs/dynamic-links/flutter/receive?hl=it.

Flutter BottomNavigationBar. URL: https://api.flutter.dev/flutter/material/BottomNavigationBar-class.html.

Flutter Deep linking. URL: https://docs.flutter.dev/ui/navigation/deep-linking.

Flutter Documentation. URL: https://docs.flutter.dev/.

Flutter Material. URL: https://docs.flutter.dev/ui/widgets/material.

Flutter Navigation and routing. URL: https://docs.flutter.dev/ui/navigation.

Flutter StreamBuilder. URL: https://api.flutter.dev/flutter/widgets/StreamBuilder-class.html.

FlutterFire. URL: https://firebase.flutter.dev/docs/overview/.

FlutterFire CLI. URL: https://firebase.flutter.dev/docs/cli/.

Manifesto Agile. URL: https://agilemanifesto.org/iso/it/manifesto.html (cit. a p. 5).

Material Design. URL: https://m3.material.io/.

Scan and Generate QR Code In Flutter. URL: https://medium.com/codechai/scan-and-generate-qr-code-in-flutter-99e4d346496b.

StatefulWidget o FutureBuilder? URL: https://andreamaglie.com/software-development/statefulwidget-o-futurebuilder/.

TableCalendar. URL: https://pub.dev/packages/table_calendar.

WAI Standards Guidelines. URL: https://www.w3.org/WAI/standards-guidelines/wcag/.