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

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 l'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, monito-

rare 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 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 con una analisi dettagliata di cosa è stato richiesto.

Il quarto capitolo approfondisce la progettazione, i *design pattern* utilizzati e la struttura del codice.

Nel quinto capitolo vengono riportate le valutazioni e le conclusioni personali del prodotto.

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.

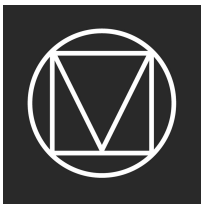

Figura 2.1: Logo del Material Design di Google

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 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

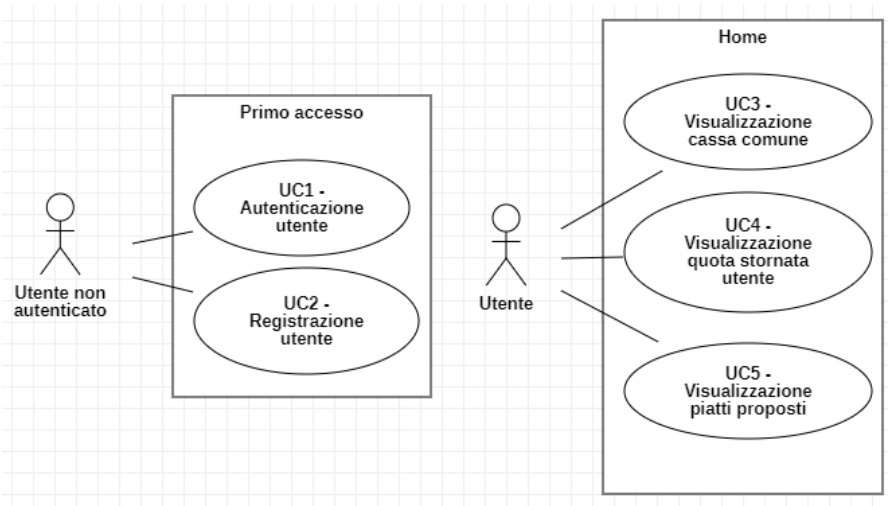

Figura 3.1: Use Case - Primo accesso e Home

UC1: Autenticazione utente

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

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*

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

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

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.

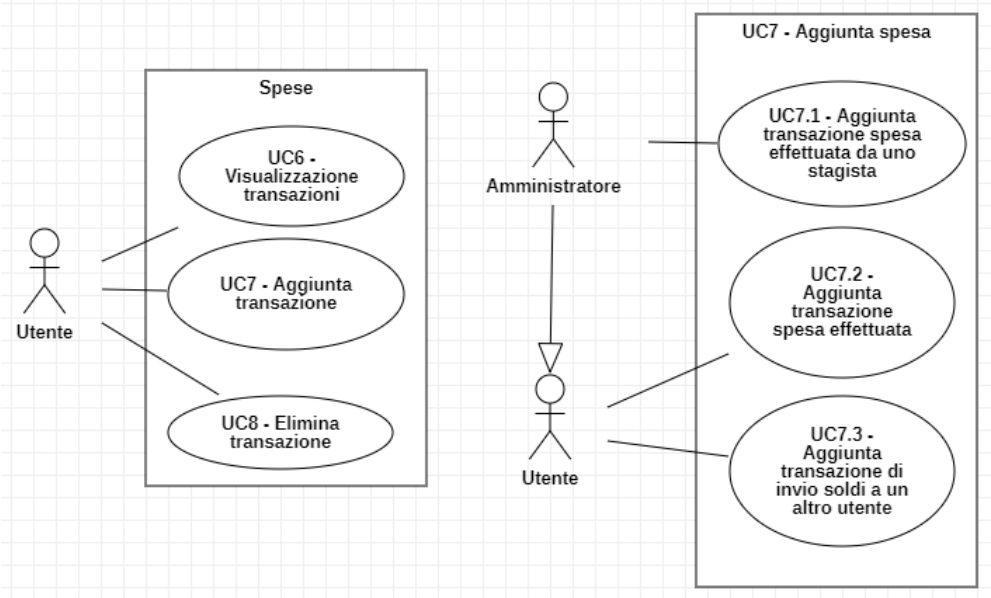

Figura 3.2: Use Case - Spese e UC7

UC6: Visualizzazione transazioni

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

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.

UC7.1: Aggiunta transazione spesa effettuata da uno stagista

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

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

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

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.

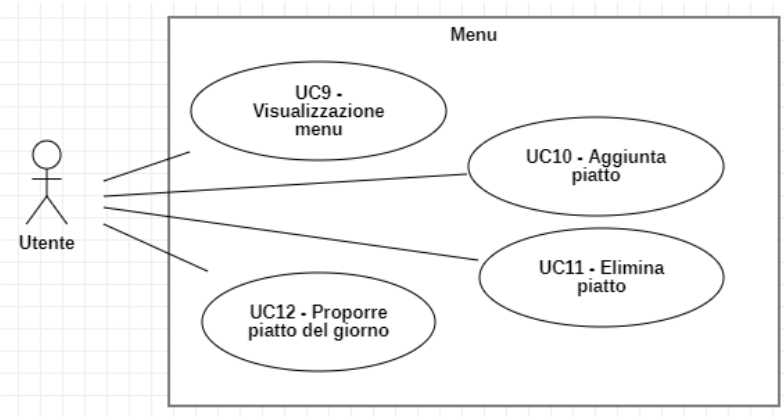

Figura 3.3: Use Case - Menu

UC9: Visualizzazione menu

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

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

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

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.

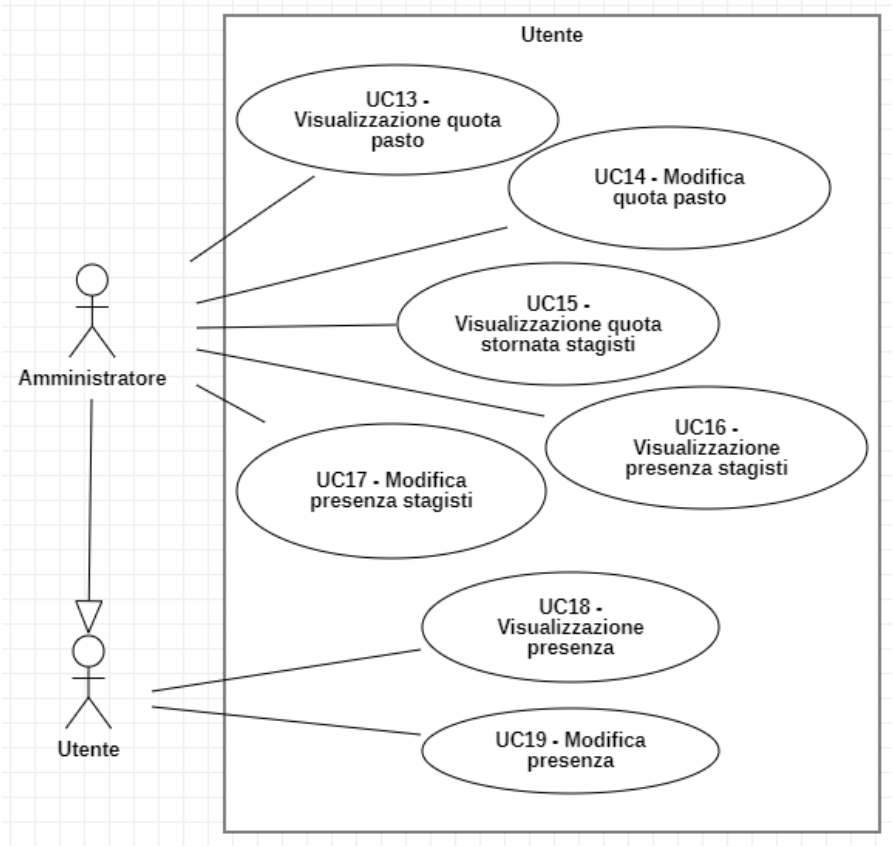

Figura 3.4: Use Case - Utente

UC13: Visualizzazione *quota pasto*[g]

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*

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

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.

UC16: Visualizzazione presenza stagisti

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

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

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

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.

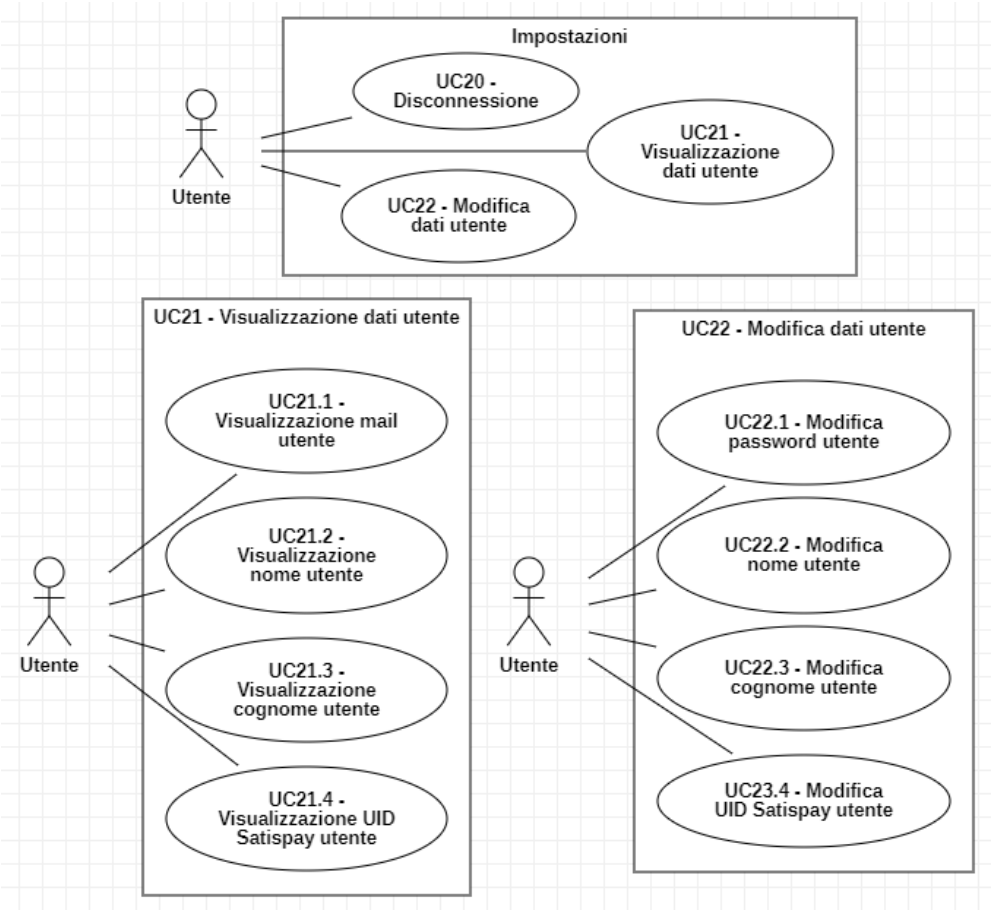

Figura 3.5: Use Case - Impostazioni, UC21 e UC22

UC20: Disconnessione

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

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.

UC21.1: Visualizzazione mail utente

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

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

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

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

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.

UC22.1: Modifica password utente

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

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

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

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.

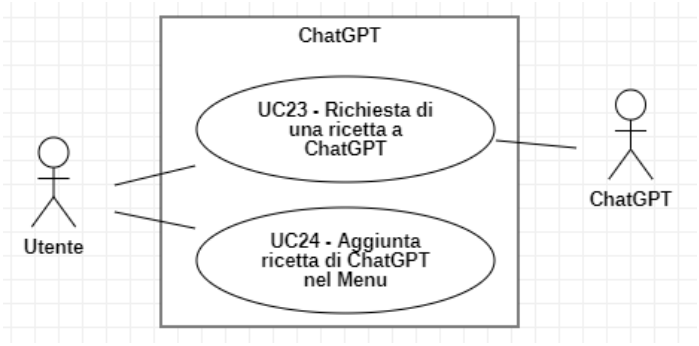

Figura 3.6: Use Case - ChatGPT

UC23: Richiesta di una ricetta a ChatGPT

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

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

3.2.1 Requisiti funzionali

Tabella 3.1: Tabella del tracciamento dei requisti funzionali dall'1 all'11

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	Si visualizza 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	Si visualizza la lista dei piatti proposti del giorno nell'app	UC5
RFO-6	Si visualizza 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

Tabella 3.2: Tabella del tracciamento dei requisti funzionali dal 12 al 35

Requisito	Descrizione	Use Case
RFO-12	Si visualizza il menu che contiene la lista dei piatti dall'app	UC9
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

3.2.2 Requisiti qualitativi

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	-

3.2.3 Requisiti di vincolo

Tabella 3.4: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Use Case
RVO-1	L'applicazione deve essere sviluppato con il *framework* Flutter	-
RVO-2	L'applicazione deve essere sviluppato 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*.

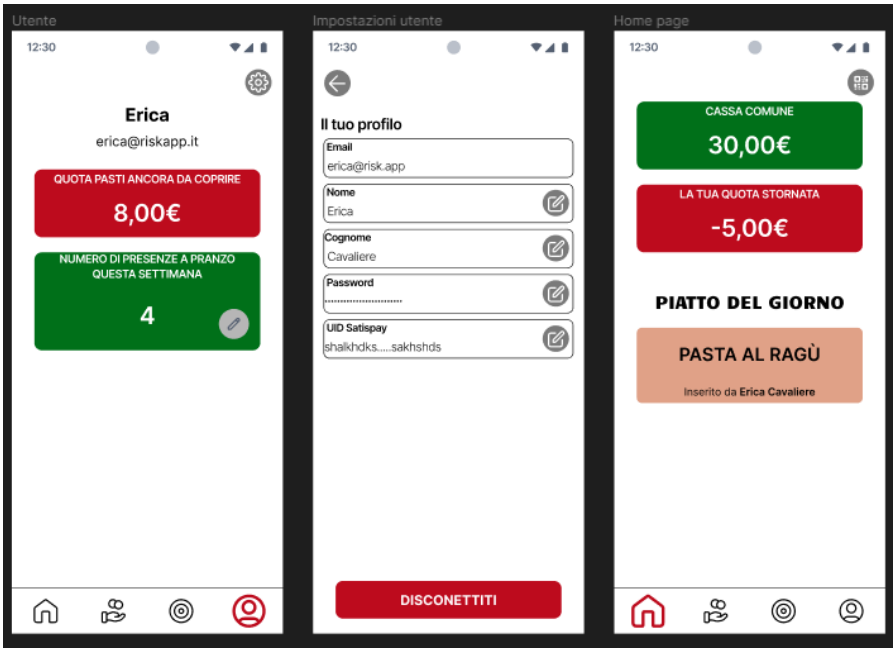

Figura 4.1: Alcune schermate progettate in Figma

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**, 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.

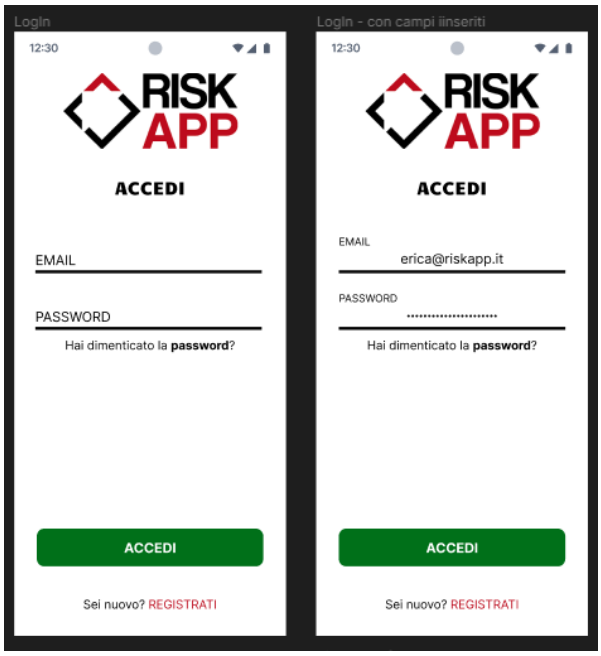

Figura 4.2: Schermata Accedi progettata in Figma

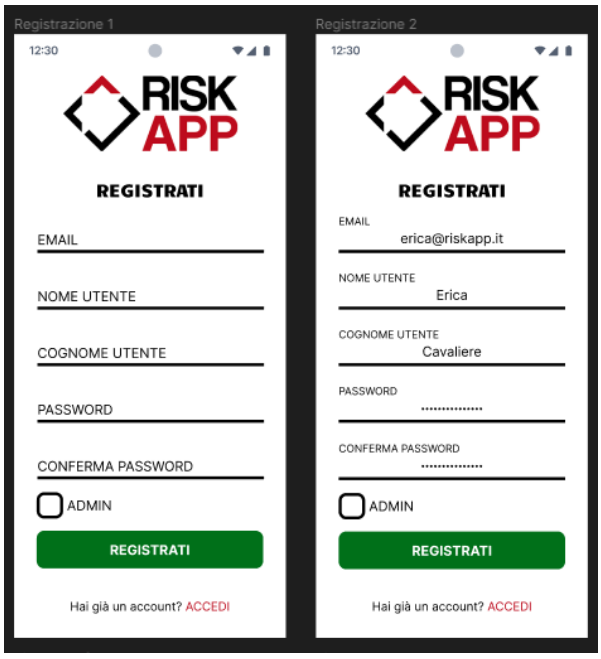

Figura 4.3: Schermata Registrati progettata in Figma

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.

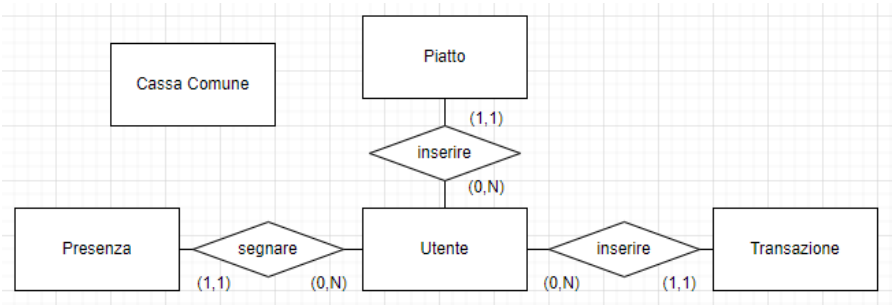

Figura 4.4: Il database progettato per l'applicazione

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)
- *quotaPasti*, di tipo number
- *quotaStornataStagisti*, di tipo number

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

utenti

- *email*, di tipo string (stringa)
- *nome*, di tipo string
- *cognome*, di tipo string
- *UID*, di tipo string; questo rappresenta il codice UID Satispay dell'utente
- *quotaStornta*, di tipo string
- *admin*, di tipo boolean (booleano)

piatti

- *nome*, di tipo string
- *ingredienti*, di tipo string
- *ricetta*, di tipo string
- *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
- *utente*, di tipo string
- *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
- *data*, di tipo timestamp
- *utente*, di tipo sting
- *stagista*, di tipo boolean; indica se è stato lo stgista 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.

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'occorente 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.

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 Acccedi 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.

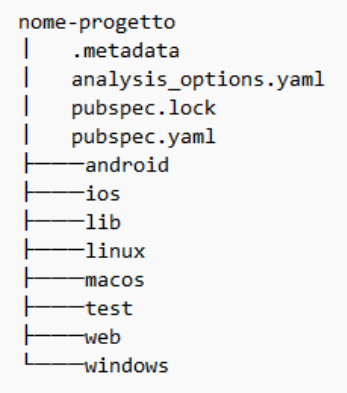

Figura 4.5: La struttura del progetto preimpostata da Flutter

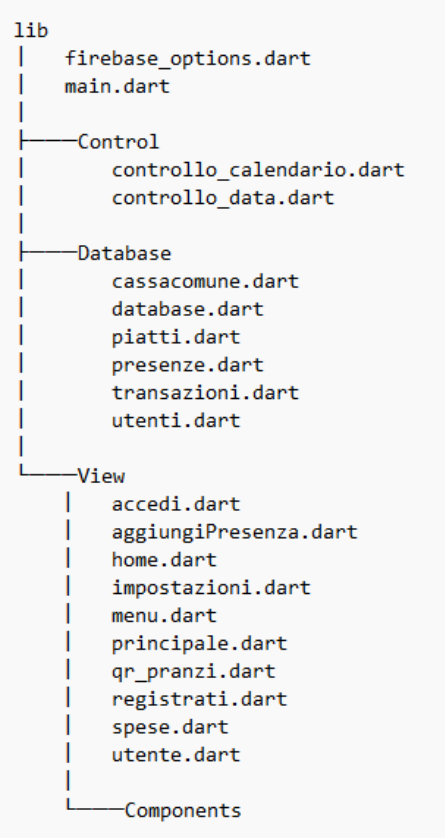

Figura 4.6: La struttura della cartella lib

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**.

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: Istruzione 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 disconnesione di un utente

Per poter lavorare con il Cloud di Firestore, è stata creata una classe per ogni entità.
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 4.13), 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*.
Non sono stati inseriti altri calendari nell'applicazione.

Una funzione fondamentale di TableCalendar è *onDaySelected*: 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*.

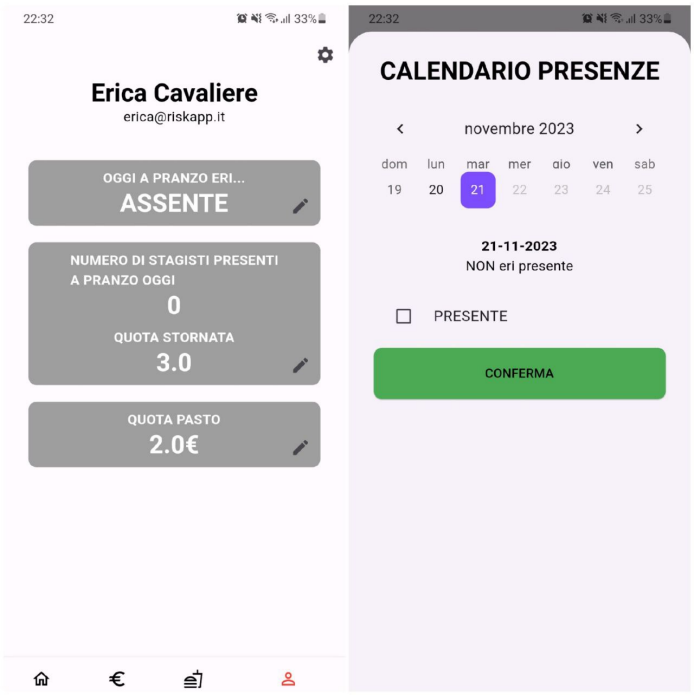

Figura 4.13: La schermata Utente e il calendario delle presenze

Figura 4.14: Le schermate Spese e Menu

4.2.4 Altre schermate dell'app

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 4.14).
Tramite la barra superiore di entrambe le schermate, è possibile aggiungere un elemento alla lista oppure utilizzare il filtro per visualizzare solo alcuni elementi specifici.

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** (*Flutter StreamBuilder*. URL: `https://api.flutter.dev/flutter/widgets/StreamBuilder-class.html`).
Dato uno *stream* di dati, lo **StreamBuilder** si occupa di creare e aggiornare dei *widget* specifici, in questo caso si occupa di aggiornare le liste presenti nelle schermate Spesa 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

4.2.6 Aprire l'app tramite QrCode

Capitolo 5

Conclusioni

Acronimi e abbreviazioni

CLI Command Line Interface. 40

IDE Integrated Development Environment. 6, 40

SDK Software Development Kit. 29, 40

UI User Interface. 3, 41

UML Unified Modeling Language. 9, 41

UX User Experience. 3, 41

WCAG Web Content Accessibility Guidelines. 4, 41

Glossario

Build indica la trasformazione del codice in un prodotto software eseguibile. 6, 7

Cassa Comune viene utilizzato questo termine per indicare i fondi dati dagli operatori aziendali per coprire i pasti. 1, 10, 21, 24, 27, 36

CLI interfaccia a riga di comando. 31, 32, 39

Componenti sono un insieme di *widget* e di elementi che insieme costituiscono un prodotto software. 4

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, 40

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, 40

IDE è un ambiente di sviluppo integrato che supporta i programmatori nello sviluppo e nel *debug* del codice. 6, 7, 39

Quota Pasto indica il quantitativo di soldi che ogni utente deve dare per ogni pranzo effettuato in azienda. 15, 22, 25, 27, 28, 34, 36

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, 10, 11, 15, 16, 21, 22, 24, 27, 36

SDK è un insieme di strumenti che consente lo sviluppo di software o firmware per una specifica piattaforma. 39

UI indica l'interfaccia grafica che viene utilizzata per le comunicazioni tra uomo e macchina. 6, 39

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. 39

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. 39

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. 39

Bibliografia

Riferimenti bibliografici

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

Siti web consultati

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.

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/.

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

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

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

Flutter StreamBuilder. URL: https://api.flutter.dev/flutter/widgets/StreamBuilder-class.html (cit. a p. 36).

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. 4).

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

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/.