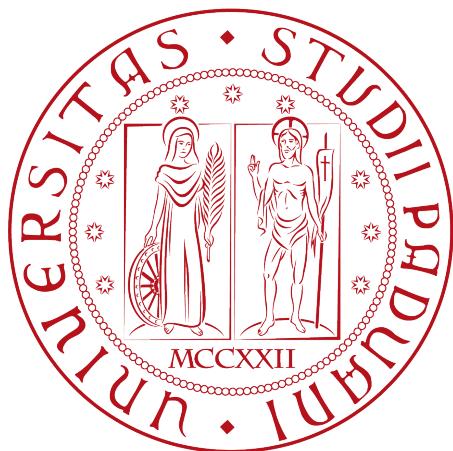


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA “TULLIO LEVI-CIVITA”

CORSO DI LAUREA IN INFORMATICA



**Progettazione e sviluppo di un'applicazione  
mobile per la gestione di Meeting Note**

*Tesi di laurea triennale*

*Relatore*

Prof. Luigi De Giovanni

*Laureando*

Sebastiano Sanson

2011880

Sebastiano Sanson: *Progettazione e sviluppo di un'applicazione mobile per la gestione di Meeting Note*, Tesi di laurea triennale, © Dicembre 2023.

# Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage avvenuto presso l'azienda RiskApp S.R.L.

Il progetto di stage consiste nella progettazione e realizzazione di un'applicazione mobile per la gestione di note. Le note sono relative a incontri tra utenti aziendali che lavorano in mobilità e clienti, e consistono in un sommario dell'incontro stesso. Lo scopo di queste note è quello di mantenere un tracciamento di tutte le trattative di una compagnia assicurativa.

Gli obiettivi da raggiungere sono molteplici. Tra quelli obbligatori sono presenti lo sviluppo dell'interfaccia utente e delle funzionalità di base: login, creazione, modifica, eliminazione e visualizzazione delle note.

Mentre, tra gli obiettivi opzionali sono presenti l'automatizzazione del processo di creazione, l'implementazione dei test per la verifica e validazione e infine l'effettuazione del *deploy* dell'applicazione sulle piattaforme *Android* e *iOS*.

*“I computer sono magnifici strumenti per la realizzazione dei nostri sogni, ma nessuna macchina può sostituire la scintilla umana di spirito, compassione, amore e comprensione”*

— Lou Gerstner

# Ringraziamenti

*Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Luigi De Giovanni, relatore della mia tesi, per la disponibilità e l'aiuto fornитomi durante la stesura del lavoro.*

*Desidero ringraziare con affetto i miei genitori per il grande aiuto e per avermi permesso di compiere questo percorso accademico. Riconosco che, a causa degli impegni e delle sfide quotidiane, i nostri rapporti umani potrebbero essersi un po' persi lungo la strada. Per questo, colgo l'occasione di dedicare questo traguardo anche a loro, per ringraziarli di cuore per tutto ciò che hanno fatto e continuano a fare per me. Spero che questo traguardo sia un motivo di orgoglio condiviso e che possiamo celebrare insieme il successo di questo percorso.*

*Desidero, infine, esprimere il mio profondo affetto anche ai miei amici, poiché la loro presenza è stata fondamentale in questo percorso. Abbiamo condiviso insieme momenti indimenticabili, vivendo gioie e affrontando sfide in questo periodo delle nostre vite, contribuendo indubbiamente alla nostra crescita personale e rafforzando il nostro legame.*

*Padova, Dicembre 2023*

Sebastiano Sanson

# Indice

<b>1 Introduzione</b>	<b>1</b>
1.1 Convenzioni tipografiche . . . . .	1
1.2 L'azienda . . . . .	1
1.3 L'idea . . . . .	2
<b>2 Il progetto di stage</b>	<b>3</b>
2.1 Descrizione del progetto . . . . .	3
2.2 Obiettivi . . . . .	3
2.2.1 Notazione . . . . .	4
2.2.2 Obiettivi fissati . . . . .	4
2.3 Pianificazione . . . . .	4
2.4 Motivazione della scelta . . . . .	6
<b>3 Analisi dei requisiti</b>	<b>7</b>
3.1 Attori . . . . .	7
3.1.1 Attori primari . . . . .	7
3.1.2 Attori secondari . . . . .	7
3.2 Token di autenticazione . . . . .	7
3.3 Casi d'uso . . . . .	7
3.4 Tracciamento dei requisiti . . . . .	30
<b>4 Progettazione</b>	<b>37</b>
4.1 Mockup . . . . .	37
4.2 Architettura . . . . .	37
4.2.1 Architettura Flutter . . . . .	37
4.2.2 Riverpod . . . . .	38
4.2.3 Architettura dell'applicazione . . . . .	39
4.3 Scelta della struttura del progetto . . . . .	40
4.4 Ambiente di sviluppo . . . . .	40
<b>5 Implementazione</b>	<b>43</b>
5.1 Struttura del progetto . . . . .	43
5.2 Components . . . . .	44
5.2.1 Alert Dialogs . . . . .	44
5.2.2 App Bars . . . . .	45
5.2.3 Biomteric Switch . . . . .	46
5.2.4 Date Picker . . . . .	46
5.2.5 Filter Panel . . . . .	47
5.2.6 Login Form . . . . .	48

5.2.7	Meeting Note Card . . . . .	49
5.2.8	Object Picker . . . . .	50
5.2.9	Recording Button . . . . .	51
5.2.10	Screens Template . . . . .	53
5.2.11	Scroll Date Picker . . . . .	54
5.2.12	Search Bar . . . . .	54
5.2.13	Text Box . . . . .	55
5.2.14	Toogle Buttons . . . . .	56
5.2.15	Warning Alert . . . . .	56
5.3	Constants . . . . .	56
5.3.1	App constants . . . . .	57
5.3.2	API constants . . . . .	57
5.4	Data . . . . .	57
5.4.1	Model . . . . .	57
5.4.2	Services . . . . .	62
5.5	Provider . . . . .	65
5.5.1	Authentication Provider . . . . .	65
5.5.2	Meeting Note Object Provider . . . . .	66
5.5.3	Meeting Note Provider . . . . .	66
5.5.4	User Provider . . . . .	67
5.6	Screens . . . . .	67
5.6.1	Account Screen . . . . .	67
5.6.2	Home Screen . . . . .	68
5.6.3	Login Screen . . . . .	70
5.6.4	Smart Creation Screen . . . . .	70
5.6.5	Wizard Screen 1 . . . . .	74
5.6.6	Wizard Screen 2 . . . . .	76
5.6.7	Wizard Screen 3 . . . . .	77
5.7	Styles . . . . .	79
5.7.1	AppColors . . . . .	79
5.7.2	ButtonStyles . . . . .	80
5.7.3	TextStyles . . . . .	81
5.7.4	AppTheme . . . . .	81
5.8	Utils . . . . .	81
5.8.1	Biometric auth . . . . .	81
5.8.2	Debouncer . . . . .	82
5.8.3	Exception handler . . . . .	82
5.8.4	Filtering . . . . .	82
5.8.5	Network handler . . . . .	82
5.8.6	User preferences . . . . .	82
5.8.7	Show alert . . . . .	83
5.8.8	Speech to text . . . . .	83
5.9	main.dart . . . . .	84
5.10	Responsività . . . . .	84
5.11	Documentazione . . . . .	85
<b>6</b>	<b>Conclusioni</b>	<b>87</b>
6.1	Obiettivi raggiunti . . . . .	87
6.2	Consuntivo delle attività . . . . .	87
6.3	Valutazione personale . . . . .	90

<i>INDICE</i>	vi
<b>Acronimi</b>	<b>91</b>
<b>Glossario</b>	<b>92</b>
<b>Riferimenti bibliografici</b>	<b>96</b>

# Elenco delle figure

3.1	Use Case - Autenticazione . . . . .	8
3.2	Use Case - Visualizzazione Meeting Note . . . . .	10
3.3	Use Case - Visualizzazione lista Meeting Note . . . . .	10
3.4	Use Case - Ricerca Meeting Note . . . . .	12
3.5	Use Case - Visualizzazione dati utente . . . . .	14
3.6	Use Case - Visualizzazione singola Meeting Note . . . . .	18
3.7	Use Case - Creazione Meeting Note . . . . .	20
3.8	Use Case - Creazione automatica . . . . .	21
3.9	Use Case - Visualizzazione risultato elaborazione . . . . .	23
3.10	Use Case - Creazione manuale (UC16) e Modifica (UC14.7) di una Meeting Note . . . . .	25
3.11	Use Case - Visualizzazione riepilogo . . . . .	28
4.1	Struttura di un'applicazione [25] in <i>Flutter</i> . . . . .	39
4.2	Esempi di approccio <i>Layer First</i> (a sinistra) e <i>Feature First</i> (a destra). . . . .	40
5.1	Struttura del progetto basato sull'approccio <i>Layer First</i> . . . . .	43
5.2	Esempi di <code>WarningAlertDialog</code> . . . . .	45
5.3	Esempio di <code>ResponseDialog</code> . . . . .	45
5.4	<code>CustomAppBar</code> senza icona (a sinistra) e <code>CustomAppBar</code> con icona (a destra)	46
5.5	<code>LoginAppBar</code> . . . . .	46
5.6	<i>Switch On-Off</i> per l'abilitazione del riconoscimento biometrico . . . . .	46
5.7	Selettore di intervallo di date . . . . .	48
5.8	Popup del <code>FilterPanel</code> . . . . .	49
5.9	<i>Meeting Note Card</i> . . . . .	51
5.10	Selettore di categoria dei clienti . . . . .	51
5.11	Pulsante nello stato di non attivazione (a sinistra) e attivazione (a destra) della dettatura vocale . . . . .	52
5.12	<code>WizardHeader</code> . . . . .	53
5.13	<code>WizardStepper</code> . . . . .	54
5.14	<code>Autocomplete</code> . . . . .	55
5.15	<code>TextBox</code> . . . . .	56
5.16	<code>ToggleButtons</code> . . . . .	56
5.17	Esempi di <code>WarningAlert</code> . . . . .	57
5.18	Account Screen . . . . .	68
5.19	Home Screen . . . . .	69
5.20	Login Screen . . . . .	70
5.21	Smart Creation Screen . . . . .	71

5.22 Confirm Creation Popup . . . . .	73
5.23 Wizard Screen 1 - Creazione (a sinistra) e Modifica (a destra) . . . . .	76
5.24 Wizard Screen 2 - Creazione (a sinistra) e Modifica (a destra) . . . . .	77
5.25 Wizard Screen 3 - Creazione (a sinistra) e Modifica (a destra) . . . . .	79
5.26 Esempi di <i>snackbar</i> . . . . .	84
6.1 Diagramma di Gantt della ripartizione definitiva delle attività durante il periodo di stage . . . . .	88

## Elenco delle tabelle

2.1 Preventivo della durata delle attività . . . . .	6
3.1 Tabella del tracciamento dei requisiti funzionali - 1 . . . . .	32
3.2 Tabella del tracciamento dei requisiti funzionali - 2 . . . . .	33
3.3 Tabella del tracciamento dei requisiti funzionali - 3 . . . . .	34
3.4 Tabella del tracciamento dei requisiti funzionali - 4 . . . . .	35
3.5 Tabella del tracciamento dei requisiti qualitativi . . . . .	35
3.6 Tabella del tracciamento dei requisiti di vincolo . . . . .	36
6.1 Tabella con gli obiettivi raggiunti . . . . .	88
6.2 Consuntivo della durata delle attività . . . . .	89

# Capitolo 1

## Introduzione

*In questo primo capitolo viene descritta brevemente l'azienda in cui si è svolto lo stage e l'idea alla base del progetto.*

### 1.1 Convenzioni tipografiche

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 i termini riportati nel glossario viene utilizzata la seguente nomenclatura: parola<sup>[gl]</sup>;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.
- per i termini salienti viene utilizzato il carattere **grassetto**;
- verrà utilizzato il seguente **font** per indicare delle parole chiave inerenti a un linguaggio di programmazione.

### 1.2 L'azienda

RiskApp S.R.L. è un'azienda, fondata nel 2015 e situata a Conselve (PD), che si occupa di fornire piattaforme digitali per la gestione del rischio nel settore assicurativo.

Il suo core business è quello fornire consulenze e sviluppare software che, attraverso la raccolta e l'analisi di dati, permette di valutare i rischi aziendali.

La principale soluzione di punta dell'azienda è la piattaforma *RiskAPP*, composta da diversi moduli, ognuno dei quali è dedicato a un servizio specifico, alcuni di questi sono:

- **monitoraggio del cliente** [52]: attraverso *Cacciatore di Dati*®, una tecnologia proprietaria che si occupa di raccogliere tutte le informazioni disponibili online sul cliente e con l'ausilio di un motore d'Intelligenza Artificiale (IA<sup>[gl]</sup>), è possibile

identificare la causa di quello che potrebbe essere un fatto precursore di un sinistro;

- **valutazione delle somme assicurative** [52]: una serie di strumenti che permettono di aiutare a effettuare una corretta valutazione delle somme assicurative, per evitare di incorrere in sovra o sottostime;
- **analisi dei rischi dovuti a calamità naturali** [52]: possibilità di prevenire conseguenze negative attraverso la valutazione dei rischi dovuti a calamità naturali;
- **analisi delle esigenze** [52]: attraverso lo strumento *Insurance Advisor*, è possibile creare un report di consulenza per il cliente, in cui vengono evidenziate le sue esigenze e le possibili soluzioni.

### 1.3 L'idea

L'idea del progetto di stage è emersa dall'esigenza, da parte dell'azienda, di adottare uno strumento digitale che permetta di tracciare gli incontri con i possibili e/o esistenti clienti<sup>[g]</sup>. Questi incontri sono molto importanti, in quanto permettono di raccogliere informazioni utili per la creazione di report di consulenza.

Dunque lo scopo finale è quello di avere un'applicazione, integrata nel *Tool trattative*, strumento incluso nella piattaforma *RiskAPP*, che permetta di raccogliere le informazioni più importanti di un incontro in una Meeting Note<sup>[g]</sup>, una nota composta da: il nome identificativo del cliente<sup>[g]</sup>, la data dell'incontro, un riassunto che ne riporta i punti salienti e l'utente che ha redatto la nota.

L'azienda, con l'ausilio di queste note, è in grado di monitorare il funnel commerciale<sup>[g]</sup>, un processo in cui le aziende trasformano normali visitatori in potenziali clienti, per poi convertirli in clienti effettivi.

Si mantiene quindi traccia di tutte le trattative di una compagnia assicurativa, in modo da poter monitorare l'andamento ed eventualmente mettere in atto delle azioni migliorative o correttive in caso di necessità.

L'applicazione è rivolta a un sottoinsieme di utenti, i commerciali di una compagnia assicurativa, che svolgono il loro lavoro in mobilità, dunque deve essere usufruibile da dispositivi *mobile*, in particolare gli *smartphone*.

Si tratta quindi di progettare e realizzare un'applicazione cross-platform<sup>[g]</sup>, utilizzabile sia su dispositivi *Android* che *iOS*.

# Capitolo 2

## Il progetto di stage

*In questo capitolo viene descritto nel dettaglio il progetto, includendo gli obiettivi richiesti e la pianificazione del lavoro per la sua realizzazione. Infine viene motivata la scelta di tale progetto.*

### 2.1 Descrizione del progetto

Lo scopo del progetto di stage è quello di sviluppare un'applicazione cross-platform<sup>[g]</sup> per la gestione di Meeting Note<sup>[g]</sup>e relativa documentazione. Nello specifico, l'applicazione deve fornire le seguenti funzionalità:

- **creazione** di una Meeting Note<sup>[g]</sup>, inserendo le informazioni richieste, essa può avvenire in due modi:
  - inserimento **manuale** dei dati attraverso un wizard<sup>[g]</sup>di creazione;
  - **automatizzazion** del processo di creazione.
- possibilità di **modificare** una Meeting Note<sup>[g]</sup>già creata, attraverso un wizard<sup>[g]</sup>di modifica, analogo a quello di creazione;
- possibilità di **eliminare** una Meeting Note<sup>[g]</sup>dalla lista di quelle create;
- **visualizzazione** di una lista di Meeting Note<sup>[g]</sup>, con la possibilità di filtrarla per cliente<sup>[g]</sup>, data e ordine di creazione (dalla più recente o meno recente), inoltre, per ciascuna, è possibile visualizzarne in dettaglio il contenuto;
- integrazione con il sistema di **autenticazione** della piattaforma *RiskAPP*, essa può avvenire in due modalità differenti:
  - inserimento **manuale** delle credenziali;
  - tramite **riconoscimento biometrico**<sup>[g]</sup>.
- possibilità di visualizzare i **dati dell'utente**.

### 2.2 Obiettivi

In questa sezione verranno illustrati gli obiettivi, definiti nel *Piano di Lavoro*, che si prevede di raggiungere durante lo svolgimento del progetto di stage.

### 2.2.1 Notazione

Si farà riferimento alla seguente notazione per indicare il grado di importanza degli obiettivi fissati:

- **O**, identifica i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
- **D**, identifica i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- **F**, identifica i requisiti facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Le sigle indicate saranno seguite da una coppia sequenziale di numeri, che identificano univocamente ogni obiettivo.

### 2.2.2 Obiettivi fissati

Si prevede lo svolgimento dei seguenti obiettivi:

- **Obbligatori**
  - **O01**: implementare un meccanismo di autenticazione attraverso delle chiamate all'API<sup>[g]</sup> della piattaforma *RiskAPP*;
  - **O02**: richiamare, attraverso richieste HTTP<sup>[g]</sup>, le API<sup>[g]</sup> per ottenere i dati necessari da visualizzare;
  - **O03**: creazione delle Meeting Note<sup>[g]</sup>;
  - **O04**: compilazione del contenuto delle Meeting Note<sup>[g]</sup> in maniera testuale e vocale;
  - **O05**: sviluppo della UI<sup>[g]</sup> grafica dell'applicazione;
- **Desiderabili**
  - **D01**: integrazione di OpenAI API<sup>[g]</sup> per la creazione automatica delle Meeting Note<sup>[g]</sup>;
  - **D02**: deploy<sup>[g]</sup> dell'applicazione sulle piattaforme *Android* e *iOS*;
- **Facoltativi**
  - **F01**: implementazione dei test per la verifica e validazione dell'applicazione.

## 2.3 Pianificazione

In questa sezione viene riportata la ripartizione settimanale delle attività da svolgere durante il periodo di stage. Tale pianificazione iniziale è stata definita anteriormente al periodo di stage e documentata nel *Piano di Lavoro*.

Nella Tabella 2.1, è stato riportato il preventivo della durata delle attività, espresso in ore, che si prevede di impiegare per la realizzazione del progetto di stage.

**Prima settimana: 24/07/23 - 28/07/23 (40 ore)**

- Incontro con il tutor aziendale per la presentazione dettagliata del progetto e comprensione dei requisiti richiesti;
- Visione del *Tool trattative* e delle API<sup>[g]</sup>, per la comprensione del contesto applicativo;
- Formazione sulle tecnologie e strumenti da utilizzare: *Dart* [14].

**Seconda settimana: 31/07/23 - 04/08/23 (40 ore)**

- Formazione sulle tecnologie e strumenti da utilizzare: *Figma* [21] e *Flutter* [23].

**Terza settimana: 07/08/23 - 11/08/23 (40 ore)**

- Realizzazione del mockup dell'applicazione;
- Analisi dei requisiti;
- Fase di progettazione dell'applicazione e realizzazione del diagramma UML delle classi;
- Stesura della documentazione relativa alle attività di analisi dei requisiti e progettazione.

**Quarta settimana: 14/08/23 - 18/08/23 (40 ore)**

- Inizio sviluppo dell'interfaccia utente (UI<sup>[g]</sup>) dell'applicazione (O05);
- Implementazione del meccanismo di autenticazione (O01);
- Chiamate alle API<sup>[g]</sup> per ottenere i dati necessari da visualizzare (O02);
- Stesura dei test per le funzionalità implementate (F01).

**Quinta settimana: 21/08/23 - 25/08/23 (40 ore)**

- Implementazione della funzionalità di creazione delle Meeting Note<sup>[g]</sup> (O03);
- Implementazione della funzionalità di compilazione delle Meeting Note<sup>[g]</sup> (O04);
- Stesura dei test per le funzionalità implementate (F01).

**Sesta settimana: 28/08/23 - 01/09/23 (40 ore)**

- Continuazione sviluppo UI<sup>[g]</sup> dell'applicazione (O05);
- Integrazione di OpenAI API<sup>[g]</sup> per la creazione automatica delle Meeting Note<sup>[g]</sup> (D01).

**Settima settimana: 04/09/23 - 08/09/23 (40 ore)**

- Completamento dello sviluppo UI<sup>[g]</sup> dell'applicazione (O05);
- Stesura dei test per le funzionalità implementate (F01).

Ore pianificate	Descrizione dell'attività
<b>80</b>	<b>Formazione sulle tecnologie</b>
<b>40</b>	<b>Definizione architettura di riferimento e relativa documentazione</b>
12	<i>Analisi del problema e del dominio applicativo</i>
22	<i>Progettazione della piattaforma e relativi test</i>
6	<i>Stesura documentazione relativa ad analisi e progettazione</i>
<b>160</b>	<b>Sviluppo</b>
26	<i>Implementazione del meccanismo di autenticazione e integrazione con il sistema preesistente per il prelevamento dei dati da visualizzare nell'applicazione</i>
34	<i>Implementazione delle funzionalità di creazione e compilazione (testuale e vocale) delle meeting-note</i>
28	<i>Integrazione del servizio OpenAI</i>
40	<i>Attività di testing</i>
32	<i>Sviluppo UI</i>
<b>40</b>	<b>Verifica e Validazione finale</b>
24	<i>Esecuzione dei test per la verifica e collaudo dell'applicazione</i>
8	<i>Stesura documentazione finale</i>
8	<i>Deploy dell'applicazione</i>
<b>Totale ore</b>	<b>320</b>

Tabella 2.1: Preventivo della durata delle attività

#### Ottava settimana: 11/09/23 - 15/09/23 (40 ore)

- Esecuzione dei test per la verifica e validazione dell'applicazione (**F01**);
- Stesura della documentazione relativa alle attività di codifica;
- Deploy<sup>[g]</sup> dell'applicazione sulle piattaforme *Android* e *iOS* (**D02**).

## 2.4 Motivazione della scelta

La scelta di tale progetto è stata dettata da varie motivazioni, alcune indipendenti da esso, ovvero la possibilità di mettersi in gioco e cercare di applicare le conoscenze acquisite durante il percorso di studi, in un contesto più professionale e reale.

Inoltre, come menzionato in precedenza, la mia esperienza pregressa con le tecnologie impiegate, mi ha semplificato la decisione, in quanto avere avuto la possibilità di utilizzarle per sviluppare un progetto in un contesto lavorativo è stato stimolante e gratificante, nonostante mi abbia anche permesso di saltare parzialmente la fase formativa, è stata comunque necessaria per approfondire le conoscenze e acquisirne di nuove.

# Capitolo 3

## Analisi dei requisiti

*In questo capitolo verranno illustrati e descritti i casi d'uso del sistema applicativo e classificati i requisiti analizzati.*

### 3.1 Attori

In questa sezione vengono definiti gli attori<sup>[g]</sup>che interagiscono con il sistema.

#### 3.1.1 Attori primari

- **Utente non autenticato:** utente che possiede le credenziali per accedere al sistema, ma non ha ancora effettuato l'autenticazione;
- **Utente autenticato:** utente che ha effettuato l'autenticazione al sistema.

#### 3.1.2 Attori secondari

- **RiskAPP API:** sistema che fornisce le API<sup>[g]</sup>del *Tool Trattative* per l'interazione con il back-end<sup>[g]</sup>dell'applicazione.

### 3.2 Token di autenticazione

Per *token* di autenticazione, del quale nel corso del documento si farà riferimento, si intende una stringa univoca che, una volta inserite e validate le credenziali, viene generata dal back-end<sup>[g]</sup>e memorizzata dall'applicazione, il cui scopo è quello, oltre al mantenimento dell'accesso al proprio account, di autenticare le richieste HTTP<sup>[g]</sup>. Si specifica inoltre che ha una durata di 4 ore, dopo le quali l'utente dovrà effettuare nuovamente l'accesso per ottenerne uno nuovo.

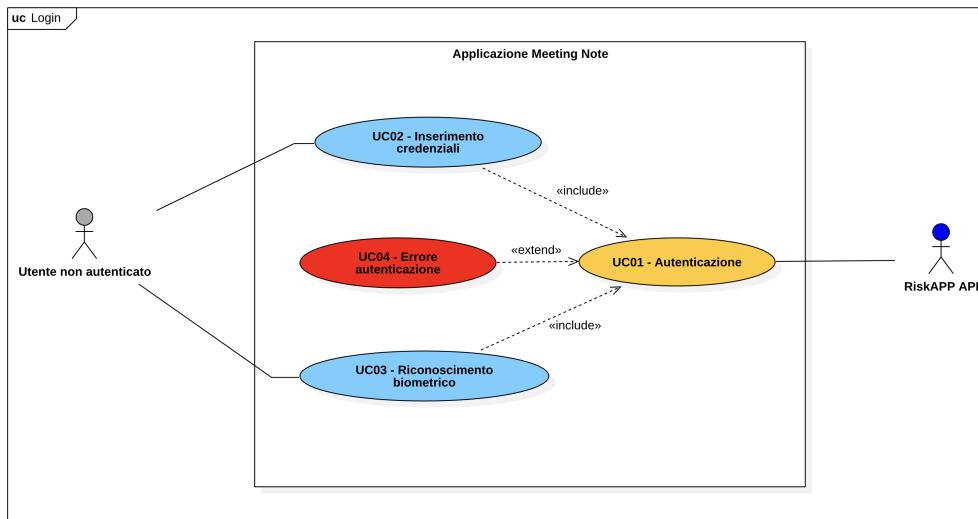
### 3.3 Casi d'uso

Per lo studio dei casi d'uso<sup>[g]</sup>del prodotto sono stati creati degli appositi diagrammi che illustrano le interazioni tra gli attori<sup>[g]</sup>e il sistema.

Ciascun caso d'uso<sup>[g]</sup>viene fornito con una descrizione che ne riporta le informazioni

principali: attori coinvolti, pre e post condizioni, scenario principale ed eventuali scenari secondari.

Per questioni di leggibilità, i diagrammi sono stati suddivisi in più parti, raggruppando i casi d'uso [g] che costituiscono una macro-funzionalità oppure specificano in dettaglio un caso d'uso [g] del sistema.



**Figura 3.1:** Use Case - Autenticazione

### UC01: Autenticazione

**Attori Principali:** Utente non autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente non è ancora autenticato nel sistema.

**Descrizione:** L'utente effettua l'autenticazione al sistema, scegliendo se farlo, inserendo manualmente le credenziali, oppure tramite riconoscimento biometrico [g].

**Postcondizioni:** L'utente è autenticato nel sistema.

**Scenario Alternativo:** Se l'autenticazione fallisce, si verifica UC04.

**Figura:** 3.1

### UC02: Inserimento credenziali

**Attori Principali:** Utente non autenticato.

**Precondizioni:** L'utente non è ancora autenticato nel sistema.

**Descrizione:** L'utente inserisce manualmente le proprie credenziali per effettuare l'autenticazione al sistema.

**Postcondizioni:** L'utente è autenticato nel sistema.

**Figura:** 3.1

### **UC03: Riconoscimento biometrico**

**Attori Principali:** Utente non autenticato.

**Precondizioni:** L'utente non è ancora autenticato nel sistema.

**Descrizione:** L'utente utilizza il riconoscimento biometrico<sup>[g]</sup> per effettuare l'autenticazione al sistema.

**Postcondizioni:** L'utente è autenticato nel sistema.

**Figura:** 3.1

### **UC04: Errore autenticazione**

**Attori Principali:** Utente non autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente non è ancora autenticato nel sistema.

**Descrizione:** L'autenticazione fallisce e l'utente viene informato dell'errore; le motivazioni possono essere le seguenti:

- le credenziali inserite non sono corrette;
- il sistema non è raggiungibile o connessione ad internet assente;
- il riconoscimento biometrico<sup>[g]</sup> è fallito.

**Postcondizioni:** L'utente non è autenticato nel sistema.

**Figura:** 3.1

### **UC05: Visualizzazione lista Meeting Note**

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente è autenticato nel sistema.

**Descrizione:** L'utente vuole visualizzare la lista di tutte le Meeting Note<sup>[g]</sup> associate.

**Postcondizioni:** È visualizzabile la lista delle Meeting Note<sup>[g]</sup>.

**Scenario Alternativo:** Se la visualizzazione fallisce, si verifica UC06.

**Figura:** 3.2

#### **UC05.1: Visualizzazione cliente delle Meeting Note**

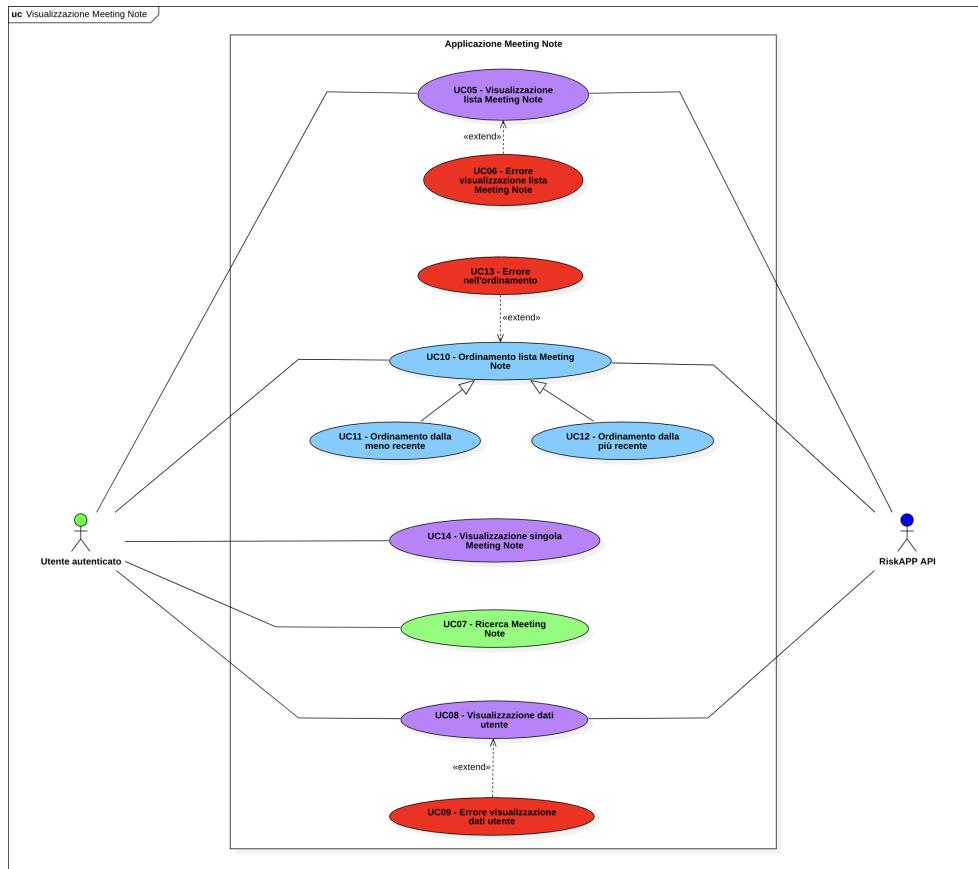
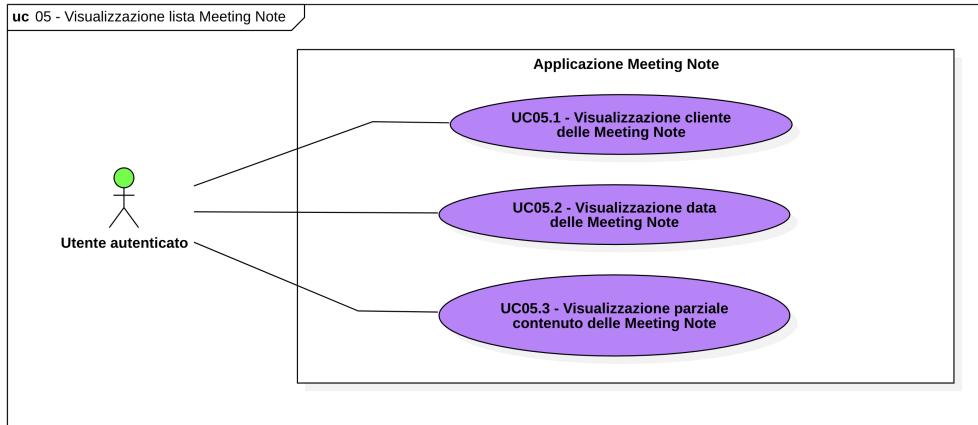
**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza la lista delle Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole visualizzare gli identificativi dei clienti<sup>[g]</sup> associati alle Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Sono visualizzabili i clienti<sup>[g]</sup> associati alle Meeting Note<sup>[g]</sup>.

**Figura:** 3.3

**Figura 3.2:** Use Case - Visualizzazione Meeting Note**Figura 3.3:** Use Case - Visualizzazione lista Meeting Note

### **UC05.2: Visualizzazione data della Meeting Note**

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza la lista delle Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole visualizzare le date degli incontri associati alle Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Sono visualizzabili le date degli incontri associati alle Meeting Note<sup>[g]</sup>.

**Figura:** 3.3

### **UC05.3: Visualizzazione parziale contenuto delle Meeting Note**

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza la lista delle Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole visualizzare parzialmente i contenuti delle Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Sono visualizzabili parzialmente i contenuti delle Meeting Note<sup>[g]</sup>.

**Figura:** 3.3

### **UC06: Errore visualizzazione lista Meeting Note**

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente è autenticato nel sistema.

**Descrizione:** La visualizzazione della lista delle Meeting Note<sup>[g]</sup> fallisce e l'utente viene informato dell'errore; le motivazioni possono essere le seguenti:

- la lista è vuota;
- il sistema non è raggiungibile;
- token di autenticazione scaduto;
- connessione ad internet assente.

**Postcondizioni:** Non è visualizzabile la lista delle Meeting Note<sup>[g]</sup>.

**Figura:** 3.2

### **UC07: Ricerca Meeting Note**

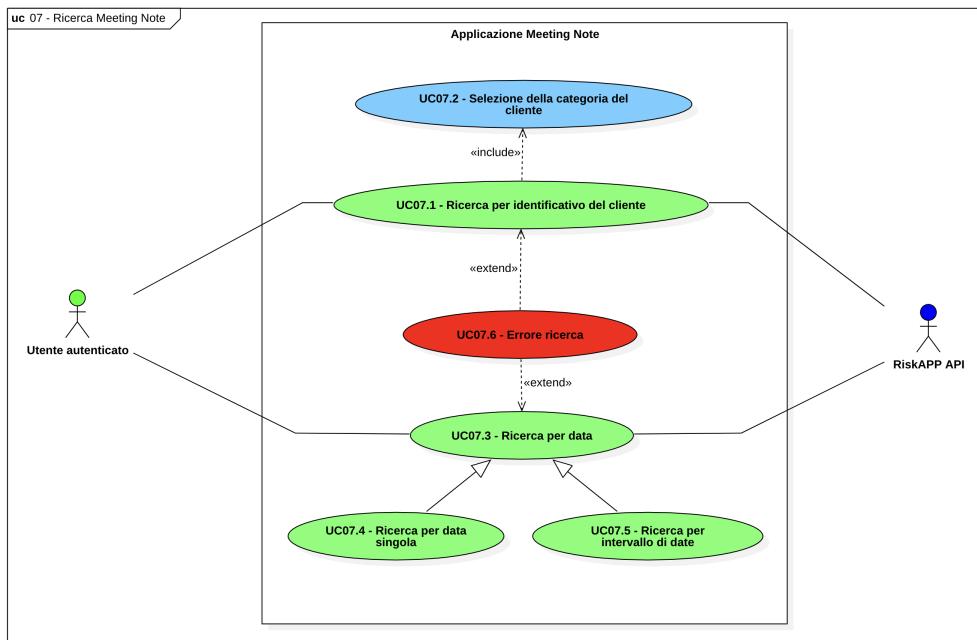
**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza la lista delle Meeting Note<sup>[g]</sup> a lui associate.

**Descrizione:** L'utente effettua una ricerca all'interno della lista delle Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Lista delle Meeting Note<sup>[g]</sup> è filtrata.

**Figura:** 3.2



**Figura 3.4:** Use Case - Ricerca Meeting Note

### UC07.1: Ricerca per identificativo del cliente

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente ha selezionato la categoria del cliente<sup>[g]</sup>.

**Descrizione:** L'utente effettua una ricerca per identificativo del cliente<sup>[g]</sup> all'interno della lista delle Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Lista delle Meeting Note<sup>[g]</sup> è filtrata per identificativo del cliente<sup>[g]</sup>.

**Scenario Alternativo:** Se la ricerca fallisce, si verifica UC07.6.

**Figura:** 3.4

### UC07.2: Selezione della categoria del cliente

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza la lista delle Meeting Note<sup>[g]</sup> a lui associate.

**Descrizione:** L'utente seleziona la categoria del cliente<sup>[g]</sup> per effettuare la ricerca per identificativo all'interno della lista delle Meeting Note<sup>[g]</sup>.

**Postcondizioni:** La categoria del cliente<sup>[g]</sup> è selezionata.

**Figura:** 3.4

### UC07.3: Ricerca per data

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API.*

**Precondizioni:** L'utente visualizza la lista delle Meeting Note<sup>[g]</sup> a lui associate.

**Descrizione:** L'utente effettua una ricerca per data all'interno della lista delle Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Lista delle Meeting Note<sup>[g]</sup> è filtrata per data.

**Scenario Alternativo:** Se la ricerca fallisce, si verifica UC07.6.

**Figura:** 3.4

#### UC07.4: Ricerca per data singola

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza la lista delle Meeting Note<sup>[g]</sup> a lui associate.

**Descrizione:** L'utente effettua una ricerca per data singola all'interno della lista delle Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Lista delle Meeting Note<sup>[g]</sup> è filtrata per data singola.

**Figura:** 3.4

#### UC07.5: Ricerca per intervallo di date

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza la lista delle Meeting Note<sup>[g]</sup> a lui associate.

**Descrizione:** L'utente effettua una ricerca per intervallo di date all'interno della lista delle Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Lista delle Meeting Note<sup>[g]</sup> è filtrata per intervallo di date.

**Figura:** 3.4

#### UC07.6: Errore ricerca

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API.*

**Precondizioni:** L'utente ha effettuato una ricerca all'interno della lista delle Meeting Note<sup>[g]</sup>.

**Descrizione:** La ricerca fallisce e l'utente viene informato dell'errore; le motivazioni possono essere le seguenti:

- la lista risultante è vuota;
- il sistema non è raggiungibile;
- token di autenticazione scaduto;
- connessione ad internet assente.

**Postcondizioni:** Lista delle Meeting Note<sup>[g]</sup> non è filtrata.

**Figura:** 3.4

### UC08: Visualizzazione dati utente

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

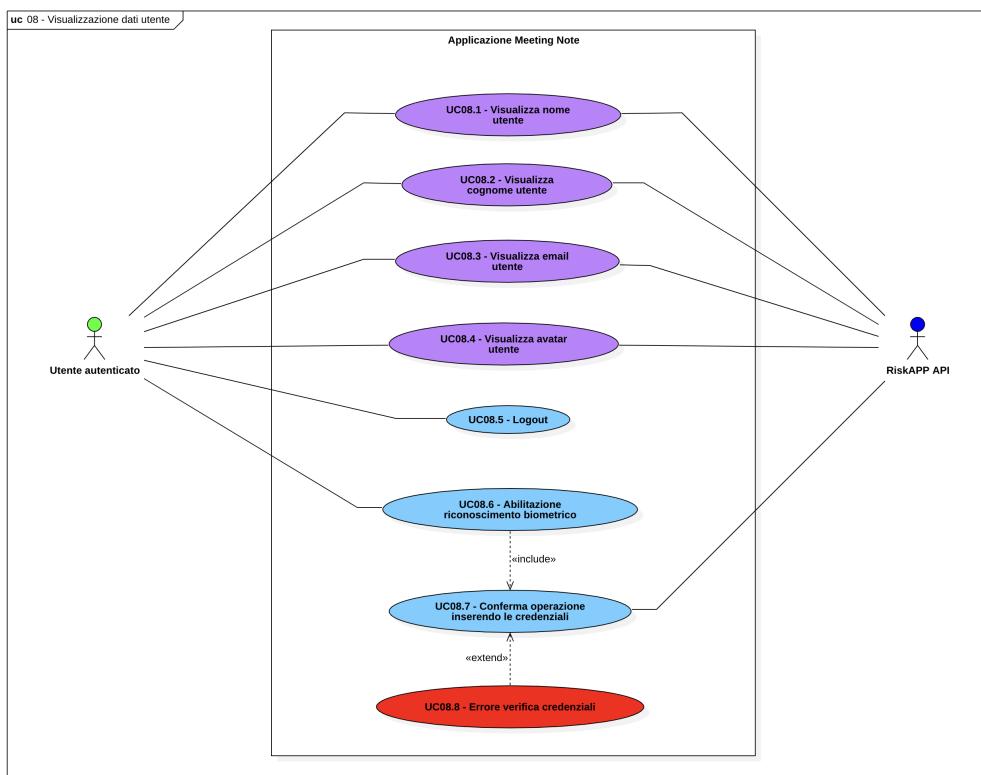
**Precondizioni:** L'utente è autenticato nel sistema.

**Descrizione:** L'utente vuole visualizzare i propri dati personali.

**Postcondizioni:** Sono visualizzabili i dati personali dell'utente.

**Scenario Alternativo:** Se la visualizzazione fallisce, si verifica UC09.

**Figura:** 3.2



**Figura 3.5:** Use Case - Visualizzazione dati utente

#### UC08.1: Visualizzazione nome utente

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente visualizza i propri dati personali.

**Descrizione:** L'utente vuole visualizzare il proprio nome.

**Postcondizioni:** È visualizzabile il nome dell'utente.

**Figura:** 3.5

**UC08.2: Visualizzazione cognome utente**

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente visualizza i propri dati personali.

**Descrizione:** L'utente vuole visualizzare il proprio cognome.

**Postcondizioni:** È visualizzabile il cognome dell'utente.

**Figura:** 3.5

**UC08.3: Visualizzazione email utente**

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente visualizza i propri dati personali.

**Descrizione:** L'utente vuole visualizzare la propria email.

**Postcondizioni:** È visualizzabile la email dell'utente.

**Figura:** 3.5

**UC08.4: Visualizzazione avatar utente**

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente visualizza i propri dati personali.

**Descrizione:** L'utente vuole visualizzare il proprio avatar.

**Postcondizioni:** È visualizzabile l'avatar dell'utente.

**Figura:** 3.5

**UC08.5: Logout**

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza i propri dati personali.

**Descrizione:** L'utente vuole effettuare il logout.

**Postcondizioni:** L'utente non è più autenticato al sistema.

**Figura:** 3.5

**UC08.6: Abilitazione riconoscimento biometrico**

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente visualizza i propri dati personali.

**Descrizione:** L'utente vuole abilitare il riconoscimento biometrico<sup>[g]</sup>, deve inserire le proprie credenziali per la conferma.

**Postcondizioni:** Riconoscimento biometrico abilitato.

**Figura:** 3.5

### UC08.7: Conferma operazione inserendo le credenziali

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente visualizza i propri dati personali.

**Descrizione:** L'utente inserisce le proprie credenziali per abilitare il riconoscimento biometrico [gl].

**Postcondizioni:** Riconoscimento biometrico abilitato.

**Scenario Alternativo:** Se la conferma fallisce, si verifica UC08.8.

**Figura:** 3.5

### UC08.8: Errore verifica credenziali

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente visualizza i propri dati personali.

**Descrizione:** La verifica delle credenziali fallisce e l'utente viene informato dell'errore; le motivazioni possono essere le seguenti:

- le credenziali inserite non sono corrette;
- il sistema non è raggiungibile;
- connessione ad internet assente.

**Postcondizioni:** Riconoscimento biometrico non abilitato.

**Figura:** 3.5

### UC09: Errore visualizzazione dati utente

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente è autenticato nel sistema.

**Descrizione:** La visualizzazione dei dati utente fallisce e l'utente viene informato dell'errore; le motivazioni possono essere le seguenti:

- il sistema non è raggiungibile;
- token di autenticazione scaduto;
- connessione ad internet assente.

**Postcondizioni:** Non sono visualizzabili i dati personali dell'utente.

**Figura:** 3.2

### UC10: Ordinamento lista Meeting Note

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente visualizza la lista delle Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole ordinare la lista delle Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Lista delle Meeting Note<sup>[g]</sup> è ordinata.

**Scenario Alternativo:** Se l'ordinamento fallisce, si verifica UC13.

**Figura:** 3.2

### UC11: Ordinamento dalla meno recente

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente visualizza la lista delle Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole ordinare la lista delle Meeting Note<sup>[g]</sup> dalla meno recente.

**Postcondizioni:** Lista delle Meeting Note<sup>[g]</sup> è ordinata dalla meno recente.

**Scenario Alternativo:** Se l'ordinamento fallisce, si verifica UC13.

**Figura:** 3.2

### UC12: Ordinamento dalla più recente

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente visualizza la lista delle Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole ordinare la lista delle Meeting Note<sup>[g]</sup> dalla più recente.

**Postcondizioni:** Lista delle Meeting Note<sup>[g]</sup> è ordinata dalla più recente.

**Scenario Alternativo:** Se l'ordinamento fallisce, si verifica UC13.

**Figura:** 3.2

### UC13: Errore nell'ordinamento

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente vuole ordinare la lista delle Meeting Note<sup>[g]</sup>.

**Descrizione:** L'ordinamento della lista delle Meeting Note<sup>[g]</sup> fallisce e l'utente viene informato dell'errore; le motivazioni possono essere le seguenti:

- il sistema non è raggiungibile;
- token di autenticazione scaduto;

- connessione ad internet assente.

**Postcondizioni:** Lista delle Meeting Note<sup>[g]</sup> non è ordinata.

**Figura:** 3.2

### UC14: Visualizzazione singola Meeting Note

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza la lista delle Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole visualizzare i dettagli di una singola Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Sono visualizzabili i dettagli di una singola Meeting Note<sup>[g]</sup>.

**Figura:** 3.2



**Figura 3.6:** Use Case - Visualizzazione singola Meeting Note

#### UC14.1: Visualizzazione cliente Meeting Note

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza i dettagli di una singola Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole visualizzare il cliente<sup>[g]</sup> di una Meeting Note<sup>[g]</sup>.

**Postcondizioni:** È visualizzabile il cliente<sup>[g]</sup> di una Meeting Note<sup>[g]</sup>.

**Figura:** 3.6

#### **UC14.2: Visualizzazione data Meeting Note**

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza i dettagli di una singola Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole visualizzare la data di una Meeting Note<sup>[g]</sup>.

**Postcondizioni:** È visualizzabile la data di una Meeting Note<sup>[g]</sup>.

**Figura:** 3.6

#### **UC14.3: Visualizzazione contenuto Meeting Note**

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza i dettagli di una singola Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole visualizzare il contenuto di una Meeting Note<sup>[g]</sup>.

**Postcondizioni:** È visualizzabile il contenuto di una Meeting Note<sup>[g]</sup>.

**Figura:** 3.6

#### **UC14.4: Visualizzazione autore Meeting Note**

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza i dettagli di una singola Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole visualizzare l'autore di una Meeting Note<sup>[g]</sup>.

**Postcondizioni:** È visualizzabile l'autore di una Meeting Note<sup>[g]</sup>.

**Figura:** 3.6

#### **UC14.5: Eliminazione della Meeting Note**

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente visualizza i dettagli di una singola Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole eliminare la Meeting Note<sup>[g]</sup> selezionata.

**Postcondizioni:** La Meeting Note<sup>[g]</sup> selezionata è stata eliminata.

**Scenario Alternativo:** Se l'eliminazione fallisce, si verifica UC14.6.

**Figura:** 3.6

#### **UC14.6: Errore nella eliminazione**

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente visualizza i dettagli di una singola Meeting Note<sup>[g]</sup>.

**Descrizione:** L'eliminazione della Meeting Note<sup>[g]</sup> selezionata fallisce e l'utente viene informato dell'errore; le motivazioni possono essere le seguenti:

- il sistema non è raggiungibile;
- token di autenticazione scaduto;
- connessione ad internet assente.

**Postcondizioni:** La Meeting Note<sup>[g]</sup>selezionata non è stata eliminata.

**Figura:** 3.6

### UC14.7: Modifica della Meeting Note

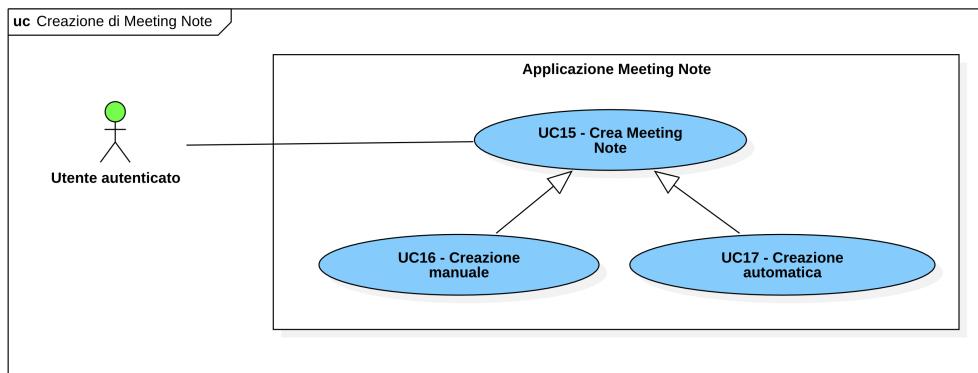
**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza i dettagli di una singola Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole modificare la Meeting Note<sup>[g]</sup>selezionata.

**Postcondizioni:** La Meeting Note<sup>[g]</sup>selezionata è stata modificata.

**Figura:** 3.6



**Figura 3.7:** Use Case - Creazione Meeting Note

### UC15: Crea Meeting Note

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente è autenticato nel sistema.

**Descrizione:** L'utente vuole creare una nuova Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Una nuova Meeting Note<sup>[g]</sup>è stata creata.

**Figura:** 3.7

### UC16: Creazione manuale

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente è autenticato nel sistema.

**Descrizione:** L'utente vuole creare manualmente una nuova Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Una nuova Meeting Note<sup>[g]</sup> è stata creata manualmente.

**Figura:** 3.7

### UC17: Creazione automatica

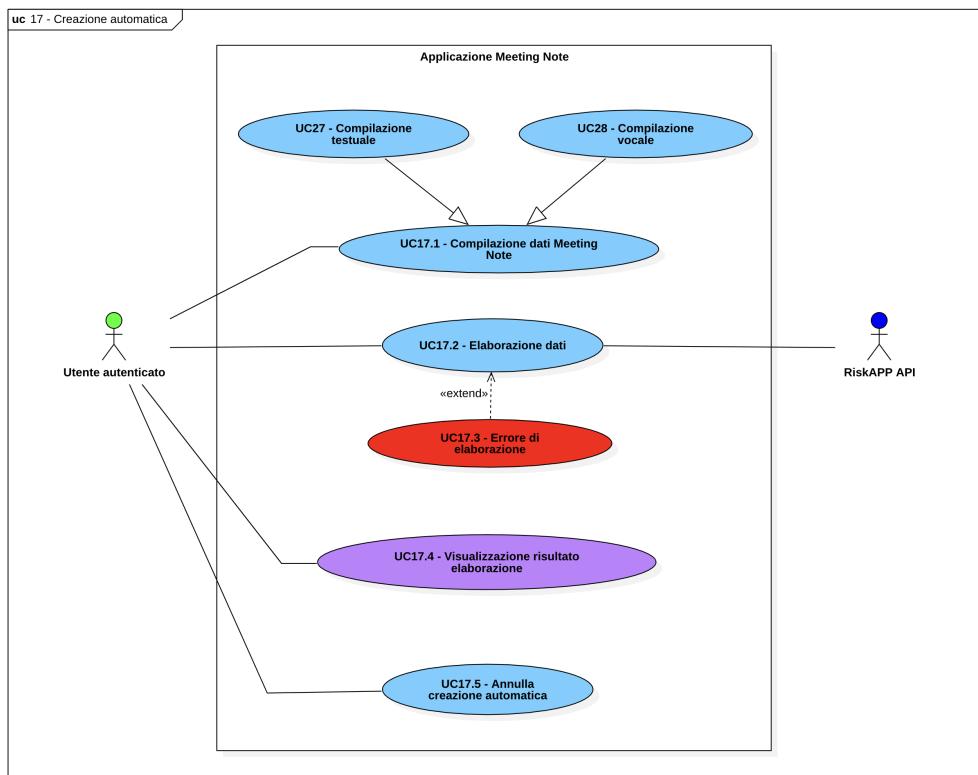
**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente è autenticato nel sistema.

**Descrizione:** L'utente vuole creare automaticamente una nuova Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Una nuova Meeting Note<sup>[g]</sup> è stata creata automaticamente.

**Figura:** 3.7



**Figura 3.8:** Use Case - Creazione automatica

### UC17.1: Compilazione dati Meeting Note

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente vuole creare automaticamente una Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente compila i dati necessari per la creazione automatica di una Meeting Note<sup>[g]</sup> sotto forma di una descrizione testuale.

**Postcondizioni:** È stato scritto una breve descrizione testuale con i dati necessari per la creazione automatica di una Meeting Note<sup>[g]</sup>.

**Figura:** 3.8

### UC17.2: Elaborazione dati

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente ha compilato i dati necessari per la creazione automatica di una Meeting Note<sup>[g]</sup>.

**Descrizione:** Il testo viene elaborato da una algoritmo di IA<sup>[s]</sup> per estrapolare i dati (cliente, data e contenuto) per la creazione automatica di una Meeting Note<sup>[g]</sup>.

**Postcondizioni:** I dati (cliente, data e contenuto) per la creazione automatica di una Meeting Note<sup>[g]</sup> sono stati estrapolati.

**Scenario Alternativo:** Se l'elaborazione fallisce, si verifica UC17.3.

**Figura:** 3.8

### UC17.3: Errore elaborazione dati

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente ha compilato i dati necessari per la creazione automatica di una Meeting Note<sup>[g]</sup>.

**Descrizione:** L'elaborazione dei dati per la creazione di una Meeting Note<sup>[g]</sup> fallisce e l'utente viene informato dell'errore; le motivazioni possono essere le seguenti:

- l'algoritmo non è stato in grado di estrarre i dati;
- il sistema non è raggiungibile;
- token di autenticazione scaduto;
- connessione ad internet assente.

**Postcondizioni:** I dati per la creazione automatica di una Meeting Note<sup>[g]</sup> non sono stati estrapolati.

**Figura:** 3.8

### UC17.4: Visualizzazione risultato elaborazione

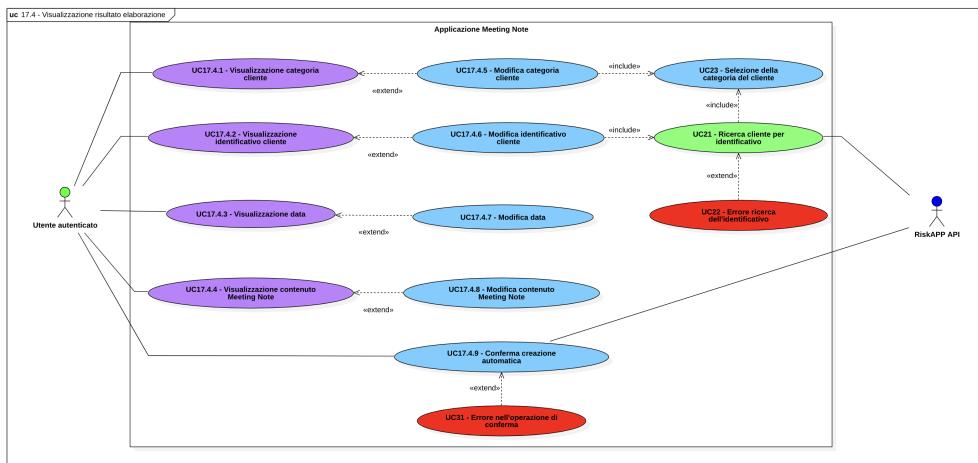
**Attori Principali:** Utente autenticato.

**Precondizioni:** L'algoritmo ha estrapolato i dati per la creazione di una Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole visualizzare i dati estrapolati dall'algoritmo per la creazione automatica di una Meeting Note<sup>[g]</sup>, per accertarsi della loro correttezza.

**Postcondizioni:** Sono visualizzabili i dati estrapolati dall'algoritmo per la creazione di una Meeting Note<sup>[g]</sup>.

**Figura:** 3.8



**Figura 3.9:** Use Case - Visualizzazione risultato elaborazione

### UC17.4.1: Visualizzazione categoria cliente

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza i dati estrapolati dall'algoritmo.

**Descrizione:** L'utente vuole visualizzare la categoria del cliente [g]estrappolato

**Postcondizioni:** È visualizzabile la categoria del cliente [g]estrappolato.

**Figura:** 3.9

### UC17.4.2: Visualizzazione identificativo cliente

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza i dati estrapolati dall'algoritmo.

**Descrizione:** L'utente vuole visualizzare del cliente [g]estrappolato

**Postcondizioni:** È visualizzabile l'identificativo del cliente [g]estrappolato.

**Figura:** 3.9

### UC17.4.3: Visualizzazione data

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza i dati estrapolati dall'algoritmo.

**Descrizione:** L'utente vuole visualizzare la data estrapolata

**Postcondizioni:** È visualizzabile la data estrapolata.

**Figura:** 3.9

### UC17.4.4: Visualizzazione il contenuto Meeting Note

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza i dati estrapolati dall'algoritmo.

**Descrizione:** L'utente vuole visualizzare il contenuto estrapolato

**Postcondizioni:** È visualizzabile il contenuto estrapolato.

**Figura:** 3.9

#### UC17.4.5: Modifica categoria cliente

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza la categoria del cliente<sup>[g]</sup>estrappolato.

**Descrizione:** L'utente modifica la categoria del cliente<sup>[g]</sup>

**Postcondizioni:** La categoria del cliente<sup>[g]</sup>è stata modificata.

**Figura:** 3.9

#### UC17.4.6: Modifica identificativo cliente

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza l'identificativo del cliente<sup>[g]</sup>estrappolato.

**Descrizione:** L'utente modifica l'identificativo del cliente<sup>[g]</sup>

**Postcondizioni:** L'identificativo del cliente<sup>[g]</sup>è stato modificato.

**Figura:** 3.9

#### UC17.4.7: Modifica data

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza la data estrapolata.

**Descrizione:** L'utente modifica la data

**Postcondizioni:** La data è stata modificata.

**Figura:** 3.9

#### UC17.4.8: Modifica contenuto Meeting Note

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza il contenuto estrapolato.

**Descrizione:** L'utente modifica il contenuto

**Postcondizioni:** Il contenuto è stato modificato.

**Figura:** 3.9

#### UC17.4.9: Conferma creazione automatica

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente vuole creare automaticamente una Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente conferma la creazione automatica di una Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Una nuova Meeting Note<sup>[g]</sup>è stata creata automaticamente.

**Scenario Alternativo:** Se la conferma fallisce, si verifica UC31.

**Figura:** 3.9

### UC17.5: Annulla creazione automatica

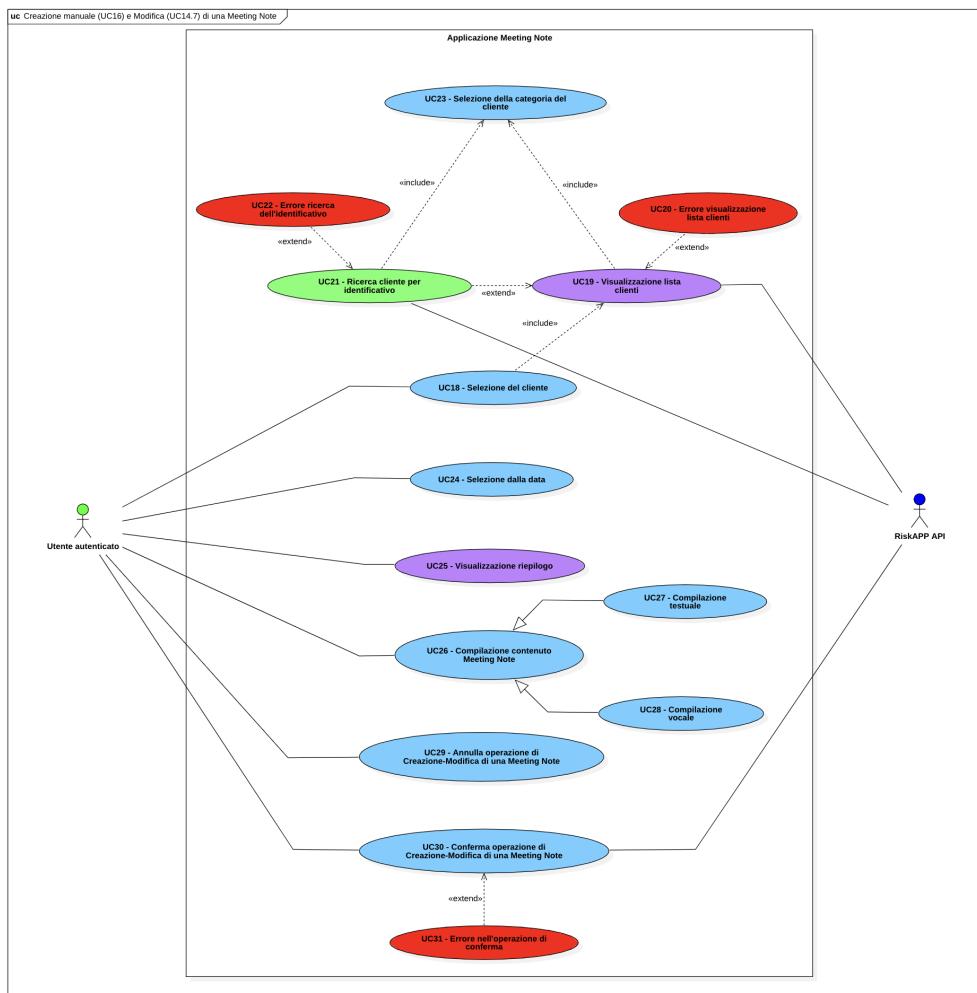
**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente è in fase di creazione automatica di una Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente annulla la creazione automatica di una Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Non è stata creata una nuova Meeting Note<sup>[g]</sup>.

**Figura:** 3.8



**Figura 3.10:** Use Case - Creazione manuale (UC16) e Modifica (UC14.7) di una Meeting Note

Per la creazione manuale (UC16) e modifica (UC14.7) di una Meeting Note<sup>[g]</sup> sono stati individuati i seguenti casi d'uso<sup>[g]</sup>, che sono in comune tra le due funzionalità sopracitate, in quanto condividono la stessa sequenza di azioni, e dunque di attori principali, secondari, precondizioni e postcondizioni.

### **UC18: Selezione del cliente**

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente vuole creare manualmente o modificare una Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente deve selezionare un cliente<sup>[g]</sup> per la creazione o la modifica di una Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Il cliente<sup>[g]</sup> è stato selezionato.

**Figura:** 3.10

### **UC19: Visualizzazione lista clienti**

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente ha selezionato la categoria del cliente<sup>[g]</sup>.

**Descrizione:** L'utente vuole visualizzare la lista di tutti i clienti<sup>[g]</sup> della categoria selezionata.

**Postcondizioni:** È visualizzabile la lista dei clienti<sup>[g]</sup> filtrata per categoria.

**Scenario Alternativo:** Se la visualizzazione fallisce, si verifica UC20.

**Figura:** 3.10

### **UC20: Errore visualizzazione lista clienti**

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente ha selezionato la categoria del cliente<sup>[g]</sup>.

**Descrizione:** La visualizzazione della lista dei clienti<sup>[g]</sup> fallisce e l'utente viene informato dell'errore; le motivazioni possono essere le seguenti:

- la lista è vuota;
- il sistema non è raggiungibile;
- token di autenticazione scaduto;
- connessione ad internet assente.

**Postcondizioni:** Non è visualizzabile la lista dei clienti<sup>[g]</sup> filtrata per categoria.

**Figura:** 3.10

### UC21: Ricerca cliente per identificativo

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente ha selezionato la categoria del cliente<sup>[g]</sup>.

**Descrizione:** L'utente effettua una ricerca per identificativo del cliente<sup>[g]</sup>.

**Postcondizioni:** L'identificativo del cliente<sup>[g]</sup> è stato trovato.

**Scenario Alternativo:** Se la ricerca fallisce, si verifica UC22.

**Figura:** 3.10

### UC22: Errore ricerca dell'identificativo

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente ha effettuato una ricerca per identificativo del cliente<sup>[g]</sup>.

**Descrizione:** La ricerca dell'identificativo del cliente<sup>[g]</sup> fallisce e l'utente viene informato dell'errore; le motivazioni possono essere le seguenti:

- la lista risultante è vuota;
- il sistema non è raggiungibile;
- token di autenticazione scaduto;
- connessione ad internet assente.

**Postcondizioni:** L'identificativo del cliente<sup>[g]</sup> non è stato trovato.

**Figura:** 3.10

### UC23: Selezione della categoria del cliente

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente è in fase di creazione o modifica.

**Descrizione:** L'utente seleziona la categoria del cliente<sup>[g]</sup> per filtrare la lista dei clienti<sup>[g]</sup> e/o la ricerca dell'identificativo.

**Postcondizioni:** La categoria del cliente<sup>[g]</sup> è stata selezionata.

**Figura:** 3.10

### UC24: Selezione della data

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente vuole creare manualmente o modificare una Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente deve selezionare una data, in cui è avvenuto l'incontro, per la creazione o la modifica di una Meeting Note<sup>[g]</sup>.

**Postcondizioni:** La data dell'incontro è stata selezionata.

**Figura:** 3.10

### UC25: Visualizzazione riepilogo

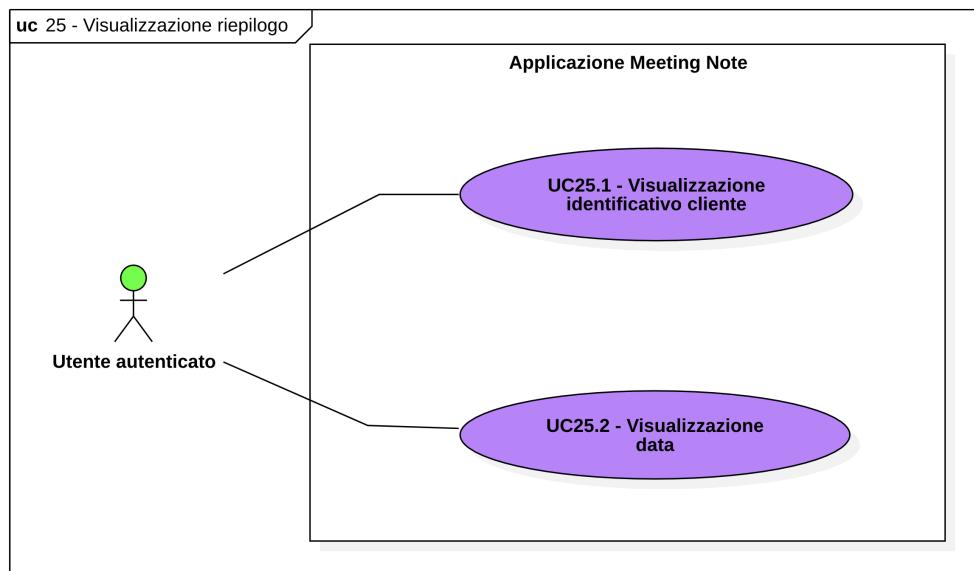
**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente ha selezionato un cliente<sup>[g]</sup> e una data.

**Descrizione:** Viene visualizzato il riepilogo dei dati selezionati.

**Postcondizioni:** È visualizzabile il riepilogo dei dati selezionati.

**Figura:** 3.10



**Figura 3.11:** Use Case - Visualizzazione riepilogo

### UC25.1: Visualizzazione identificativo cliente

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza il riepilogo.

**Descrizione:** Viene visualizzato l'identificativo del cliente<sup>[g]</sup> selezionato.

**Postcondizioni:** È visualizzabile l'identificativo del cliente<sup>[g]</sup> selezionato.

**Figura:** 3.11

### UC25.2: Visualizzazione data

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente visualizza il riepilogo.

**Descrizione:** Viene visualizzata la data selezionata.

**Postcondizioni:** È visualizzabile la data selezionata.

**Figura:** 3.11

### UC26: Compilazione contenuto Meeting Note

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente vuole creare manualmente o modificare una Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente deve compilare il contenuto della Meeting Note<sup>[g]</sup>.

**Postcondizioni:** Il contenuto della Meeting Note<sup>[g]</sup> è compilato.

**Figura:** 3.10

### UC27: Compilazione testuale

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente deve compilare il contenuto della Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente deve compilare il contenuto della Meeting Note<sup>[g]</sup> con l'ausilio della tastiera.

**Postcondizioni:** Il contenuto della Meeting Note<sup>[g]</sup> è stato compilato con l'ausilio della tastiera.

**Figura:** 3.10

### UC28: Compilazione vocale

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente deve compilare il contenuto della Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente deve compilare il contenuto della Meeting Note<sup>[g]</sup> attraverso la dettatura vocale.

**Postcondizioni:** Il contenuto della Meeting Note<sup>[g]</sup> è stato compilato attraverso la dettatura vocale.

**Figura:** 3.10

### UC29: Annulla operazione di Creazione-Modifica di una Meeting Note

**Attori Principali:** Utente autenticato.

**Precondizioni:** L'utente ha selezionato: cliente<sup>[g]</sup>, data e compilato il contenuto della Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole annullare l'operazione di creazione o modifica di una Meeting Note<sup>[g]</sup>.

**Postcondizioni:** L'operazione di creazione o modifica di una Meeting Note<sup>[g]</sup> è annullata.

**Figura:** 3.10

### UC30: Conferma operazione di Creazione-Modifica di una Meeting Note

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente ha selezionato: cliente<sup>[g]</sup>, data e compilato il contenuto della Meeting Note<sup>[g]</sup>.

**Descrizione:** L'utente vuole confermare l'operazione di creazione o modifica di una Meeting Note<sup>[g]</sup>.

**Postcondizioni:** L'operazione di creazione o modifica di una Meeting Note<sup>[g]</sup> è confermata.

**Scenario Alternativo:** Se la conferma fallisce, si verifica UC31.

**Figura:** 3.10

### UC31: Errore nell'operazione di conferma

**Attori Principali:** Utente autenticato.

**Attori Secondari:** *RiskAPP API*.

**Precondizioni:** L'utente ha confermato l'operazione di creazione o modifica di una Meeting Note<sup>[g]</sup>.

**Descrizione:** La conferma dell'operazione di creazione o modifica della Meeting Note<sup>[g]</sup> fallisce e l'utente viene informato dell'errore; le motivazioni possono essere le seguenti:

- il sistema non è raggiungibile;
- token di autenticazione scaduto;
- connessione ad internet assente.

**Postcondizioni:** La conferma dell'operazione di creazione o modifica di una Meeting Note<sup>[g]</sup> è fallita.

**Figura:** 3.10

## 3.4 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e dei casi d'uso effettuata sul progetto è stata stilata la tabella che traccia i requisiti.

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)(N/D/O) dove:

R = requisito

F = funzionale

Q = qualitativo

V = di vincolo

N = obbligatorio (necessario)

D = desiderabile

Z = opzionale

Di seguito sono riportate le tabelle che raccolgono, per tipologia, i requisiti individuati.

- **Requisiti funzionali:** descrivono le funzionalità offerte dal prodotto software. Si faccia riferimento alle Tabelle 3.1, 3.2, 3.3, 3.4;
- **Requisiti qualitativi:** descrivono le caratteristiche qualitative che il prodotto software deve possedere. Si faccia riferimento alla Tabella 3.5;
- **Requisiti di vincolo:** descrivono i vincoli che il prodotto software deve rispettare. Si faccia riferimento alla Tabella 3.6.

Nelle tabelle relative ai requisiti funzionali, alcuni di questi coinvolgono più casi d'uso [\[g\]](#), la motivazione risiede nel fatto che essi ne modellano un comportamento comune, nello specifico si tratta di casi in cui vengono descritti scenari alternativi per la gestione di eccezioni.

Di seguito sono elencati, insieme ai relativi requisiti, i casi d'uso [\[g\]](#) che, per questioni di leggibilità, non sono stati inseriti nelle tabelle:

- **RFN-4 :** UC08.8;
- **RFN-5 :** UC06, UC07.6, UC08.8, UC09, UC13, UC14.6, UC17.3, UC20, UC22, UC31;
- **RFN-7:** UC06, UC07.6, UC08.8, UC09, UC13, UC14.6, UC17.3, UC20, UC22, UC31;
- **RFN-13:** UC07.6, UC08.8, UC09, UC13, UC14.6, UC17.3, UC20, UC22, UC31;
- **RFN-64:** UC22.

**Tabella 3.1:** Tabella del tracciamento dei requisiti funzionali - 1

<b>Requisito</b>	<b>Descrizione</b>	<b>Use Case</b>
RFN-1	Il sistema permette di effettuare l'autenticazione	UC01
RFN-2	Il sistema permette di inserire le credenziali ( <i>username</i> e <i>password</i> ) per effettuare l'autenticazione	UC02
RFN-3	Il sistema permette il riconoscimento biometrico per effettuare l'autenticazione	UC03
RFN-4	Il sistema deve notificare l'utente in caso di inserimento di credenziali errate	UC04
RFN-5	Il sistema deve notificare l'utente in caso in cui il sistema, ovvero la piattaforma <i>RiskAPP</i> non sia raggiungibile	UC04
RFN-6	Il sistema deve permettere all'utente, in caso in cui il riconoscimento biometrico fallisca, di poter inserire le credenziali manualmente	UC04
RFN-7	Il sistema deve notificare l'utente in caso di connessione internet assente	UC04
RFN-8	Il sistema permette di visualizzare la lista di <i>Meeting Note</i>	UC05
RFN-9	Il sistema permette di visualizzare i clienti di ciascuna <i>Meeting Note</i> nella lista	UC05.1
RFN-10	Il sistema permette di visualizzare la data dell'incontro di ciascuna <i>Meeting Note</i> nella lista	UC05.2
RFN-11	Il sistema permette di visualizzare parzialmente il contenuto di ciascuna <i>Meeting Note</i> nella lista	UC05.3
RFN-12	Il sistema deve notificare l'utente in caso la lista di <i>Meeting Note</i> sia vuota	UC06
RFN-13	Il sistema deve notificare l'utente in caso sia scaduto il token di autenticazione	UC06
RFN-14	Il sistema permette di effettuare una ricerca nella lista di <i>Meeting Note</i>	UC07
RFN-15	Il sistema permette di effettuare una ricerca nella lista di <i>Meeting Note</i> per identificativo del cliente	UC07.1
RFN-16	Il sistema permette di selezionare la categoria del cliente in modo da effettuare una ricerca nella lista di <i>Meeting Note</i> per identificativo del cliente	UC07.2
RFN-17	Il sistema permette di effettuare una ricerca nella lista di <i>Meeting Note</i> per data	UC07.3
RFN-18	Il sistema permette di effettuare una ricerca nella lista di <i>Meeting Note</i> per data singola	UC07.4
RFN-19	Il sistema permette di effettuare una ricerca nella lista di <i>Meeting Note</i> per intervallo di date	UC07.5
RFN-20	Il sistema permette di notificare l'utente in caso la lista filtrata sia vuota	UC07.6
RFN-21	Il sistema permette di visualizzare i dati personali dell'utente	UC08

**Tabella 3.2:** Tabella del tracciamento dei requisiti funzionali - 2

Requisito	Descrizione	Use Case
RFN-22	Il sistema permette di visualizzare il nome dell'utente	UC08.1
RFN-23	Il sistema permette di visualizzare il cognome dell'utente	UC08.2
RFN-24	Il sistema permette di visualizzare la email dell'utente	UC08.3
RFN-25	Il sistema permette di visualizzare l'avatar dell'utente	UC08.4
RFN-26	Il sistema permette di effettuare il logout	UC08.5
RFN-27	Il sistema permette di abilitare il riconoscimento biometrico	UC08.6
RFN-28	Il sistema permette di confermare l'abilitazione del riconoscimento biometrico, inserendo le credenziali ( <i>username</i> e <i>password</i> )	UC08.7
RFN-29	Il sistema permette di notificare l'utente in caso in cui la conferma per l'abilitazione del riconoscimento biometrico non vada a buon fine	UC08.8
RFN-30	Il sistema permette di notificare l'utente se la visualizzazione dei dati personali fallisce	UC09
RFN-31	Il sistema permette di ordinare la lista di <i>Meeting Note</i>	UC10
RFN-32	Il sistema permette di ordinare la lista di <i>Meeting Note</i> per data meno recente	UC11
RFN-33	Il sistema permette di ordinare la lista di <i>Meeting Note</i> per data più recente	UC12
RFN-34	Il sistema permette di notificare l'utente se l'ordinamento della lista di <i>Meeting Note</i> fallisce	UC13
RFN-35	Il sistema permette di visualizzare il contenuto integrale di una <i>Meeting Note</i> selezionata	UC14
RFN-36	Il sistema permette di visualizzare l'identificativo del cliente di una <i>Meeting Note</i> selezionata	UC14.1
RFN-37	Il sistema permette di visualizzare la data dell'incontro di una <i>Meeting Note</i> selezionata	UC14.2
RFN-38	Il sistema permette di visualizzare il contenuto dell'incontro di una <i>Meeting Note</i> selezionata	UC14.3
RFN-39	Il sistema permette di visualizzare l'autore di una <i>Meeting Note</i> selezionata	UC14.4
RFN-40	Il sistema permette di eliminare una <i>Meeting Note</i> selezionata	UC14.5
RFN-41	Il sistema permette di notificare l'utente in caso in cui l'eliminazione di una <i>Meeting Note</i> selezionata fallisca	UC14.6
RFN-42	Il sistema permette di modificare una <i>Meeting Note</i> selezionata	UC14.7

**Tabella 3.3:** Tabella del tracciamento dei requisiti funzionali - 3

<b>Requisito</b>	<b>Descrizione</b>	<b>Use Case</b>
RFN-43	Il sistema permette di effettuare la creazione di una <i>Meeting Note</i>	UC15
RFN-44	Il sistema permette di effettuare la creazione manuale di una <i>Meeting Note</i>	UC16
RFD-45	Il sistema permette di effettuare la creazione automatica di una <i>Meeting Note</i>	UC17
RFD-46	Il sistema permette di compilare i dati (cliente, data e contenuto), sotto forma di una descrizione testuale, di una <i>Meeting Note</i> per la creazione automatica	UC17.1
RFD-47	Il sistema permette di effettuare l'elaborazione del testo da parte di un algoritmo di IA [gl] per l'estrapolazione dei dati (cliente, data e contenuto)	UC17.2
RFD-48	Il sistema permette di notificare l'utente in caso in cui l'elaborazione del testo non vada a buon fine	UC17.3
RFD-49	Il sistema permette di notificare l'utente in caso in cui l'algoritmo di elaborazione non è in grado di estrarre, dal testo, i dati (cliente, data e contenuto)	UC17.3
RFD-50	Il sistema permette di visualizzare il risultato dell'elaborazione dei dati	UC17.4
RFD-51	Il sistema permette di visualizzare la categoria del cliente estrapolato	UC17.4.1
RFD-52	Il sistema permette di visualizzare l'identificativo del cliente estrapolato	UC17.4.2
RFD-53	Il sistema permette di visualizzare la data dell'incontro estrapolata	UC17.4.3
RFD-54	Il sistema permette di visualizzare il contenuto dell'incontro estrapolato	UC17.4.4
RFD-55	Il sistema permette di modificare la categoria del cliente estrapolato	UC17.4.5
RFD-56	Il sistema permette di modificare l'identificativo del cliente estrapolato	UC17.4.6
RFD-57	Il sistema permette di modificare la data dell'incontro estrapolata	UC17.4.7
RFD-58	Il sistema permette di modificare il contenuto dell'incontro estrapolato	UC17.4.8
RFD-59	Il sistema permette di confermare la creazione automatica	UC17.4.9
RFD-60	Il sistema permette di annullare l'operazione di creazione automatica	UC17.5

**Tabella 3.4:** Tabella del tracciamento dei requisiti funzionali - 4

Requisito	Descrizione	Use Case
RFN-61	Il sistema permette di selezionare il cliente per la creazione manuale di una <i>Meeting Note</i>	UC18
RFN-62	Il sistema permette di visualizzare la lista dei clienti per la selezione	UC19
RFN-63	Il sistema permette di notificare l'utente in caso cui la visualizzazione della lista dei clienti fallisca	UC20
RFN-64	Il sistema deve notificare l'utente in caso la lista dei clienti sia vuota	UC20
RFN-65	Il sistema deve permettere di ricercare un cliente nella lista per identificativo	UC21
RFN-66	Il sistema deve permettere di selezionare la categoria del cliente	UC23
RFN-67	Il sistema permette di selezionare la data per la creazione manuale di una <i>Meeting Note</i>	UC24
RFN-68	Il sistema permette di visualizzare il riepilogo dei dati selezionati	UC25
RFN-69	Il sistema permette di visualizzare l'identificativo del cliente selezionato	UC25.1
RFN-70	Il sistema permette di visualizzare la data selezionata	UC25.2
RFN-71	Il sistema permette di compilare il contenuto della <i>Meeting Note</i>	UC26
RFN-72	Il sistema permette di compilare il contenuto della <i>Meeting Note</i> con l'ausilio della tastiera	UC27
RFN-73	Il sistema permette di compilare il contenuto della <i>Meeting Note</i> con l'ausilio della dettatura vocale	UC28
RFN-74	Il sistema permette di annullare l'operazione di creazione o modifica di una <i>Meeting Note</i>	UC29
RFN-75	Il sistema permette di confermare l'operazione di creazione o modifica di una <i>Meeting Note</i>	UC30
RFN-76	Il sistema permette di notificare l'utente in caso in cui la conferma dell'operazione di creazione o modifica di una <i>Meeting Note</i> non vada a buon fine	UC31

**Tabella 3.5:** Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Fonte
RQD-1	Il sistema deve garantire il corretto funzionamento attraverso l'implementazione ed esecuzione di test	F01
RQN-2	Il codice prodotto deve essere disponibile su GitHub <sup>[g]</sup> , nella repository aziendale	Azienda
RQN-3	Il codice prodotto deve essere documentato	Azienda
RQN-4	L'applicazione deve soddisfare la proprietà di responsività <sup>[g]</sup> del layout	Azienda

**Tabella 3.6:** Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Fonte
RVN-1	Utilizzo del linguaggio <i>dart</i> [14] per lo sviluppo dell'applicazione	Azienda
RVN-2	Utilizzo di <i>Flutter</i> [23] per lo sviluppo dell'applicazione	Azienda
RVN-3	Utilizzo delle API <sup>[g]</sup> della piattaforma <i>RiskAPP</i> per la comunicazione con il back-end <sup>[g]</sup>	Azienda
RVN-4	Assicurare la compatibilità con gli <i>smartphone</i>	Azienda
RVN-5	Assicurare la compatibilità con la versione del sistema operativo <i>Android</i> $\geq 5$	RVN-2
RVN-6	Assicurare la compatibilità con la versione del sistema operativo <i>iOS</i> $\geq 11$	RVN-2
RVZ-7	Eseguire il deploy dell'applicazione sui sistemi operativi <i>Android</i> e <i>iOS</i>	D02
RVN-8	Per l'implementazione della dettatura vocale, utilizzare una libreria di terze parti opportuna	RFN-73
RVN-9	Per l'implementazione del riconoscimento biometrico, utilizzare una libreria di terze parti opportuna	RFN-3
RVN-10	Assicurare il salvataggio del token di autenticazione nella memoria locale del dispositivo	RFN-1
RVN-11	Implementare una libreria di terze parti per il salvataggio in locale del token di autenticazione	RVN-10
RVN-12	Assicurare il salvataggio e la cifratura delle credenziali di accesso nella memoria locale del dispositivo	RFN-2
RVN-13	Implementare una libreria di terze parti per la cifratura delle credenziali di accesso	RVN-12
RVN-14	Assicurare la presenza di connessione ad internet per la comunicazione con le API <sup>[g]</sup> della piattaforma <i>RiskAPP</i>	RVN-3
RVN-15	Implementare una libreria di terze parti per effettuare le chiamate HTTP <sup>[g]</sup>	RVN-3
RVN-16	Tutte le <i>Meeting Note</i> create devono essere salvate nel <i>backend</i>	Azienda

# Capitolo 4

# Progettazione

In questo capitolo verrà illustrata la fase di progettazione del prodotto, partendo dalla realizzazione di un mockup, passando per la definizione dell'architettura e le tecnologie da utilizzare.

## 4.1 Mockup

Il primo passo per la progettazione dell'applicazione è stato quello di realizzare un mockup<sup>[8]</sup>, con lo scopo di definirne il più dettagliatamente possibile l'interfaccia grafica: il numero di viste necessarie, la loro struttura, i componenti grafici e la loro disposizione, la *palette* dei colori, come l'utente interagirà con l'applicazione e come questa dovrà rispondere a tali interazioni, simulabili attraverso un prototipo. Infatti, per l'implementazione della UI<sup>[8]</sup>, si è rivelato essere di notevole utilità, in quanto ha reso più semplice e veloce la realizzazione di quest'ultima, avendo compreso a priori come questa dovesse essere strutturata.

Inoltre, il mockup è stato utilizzato per definire le funzionalità che l'applicazione deve offrire, in modo da avere un'idea più chiara di come queste debbano essere implementate, ed infine, è stato indispensabile per la fase di *analisi dei requisiti*.

Si specifica che nel corso della fase di implementazione sono state apportate delle modifiche all'interfaccia grafica per migliorarne l'usabilità, mantenendo però invariata la struttura generale dell'applicazione e prestando particolare attenzione all'esperienza utente, in modo da rendere l'utilizzo dell'applicazione, in mobilità, il più semplice e intuitivo possibile.

## 4.2 Architettura

### 4.2.1 Architettura Flutter

*Flutter* è un framework<sup>[g]</sup> che fornisce una struttura completa, comprensiva di una serie di strumenti e librerie, facilitando lo sviluppo di applicazioni *cross-platform*. Queste applicazioni sono progettate per essere eseguite su vari sistemi operativi, tra cui *Android*, *iOS*, per dispositivi *mobile* e *MacOS*, *Linux* e *Windows* per dispositivi *desktop*.

Per comprendere al meglio questa peculiarità di *Flutter*, è necessario analizzare l'archi-

tettura [25] delle applicazioni realizzate con esso, che sono composte dagli elementi illustrati nella Figura 4.1, tra i quali, i principali sono descritti di seguito:

- **Embedder:** fornisce un punto d'ingresso con il sistema operativo ospitante per accedere ai servizi forniti da esso. Questo consente dunque l'esecuzione dell'applicazione su diverse piattaforme e la possibilità di utilizzare librerie per accedere a funzionalità esclusive di determinati sistemi operativi, oppure viceversa, integrare il codice *Flutter* in un'applicazione nativa già esistente;
- **Flutter engine:** è il componente centrale che fornisce l'implementazione a basso livello dell'API<sup>[g]</sup> principale di *Flutter*, includendo la grafica, il layout di testo, operazioni di *I/O*, rete, ecc.;
- **Flutter framework:** componente con il quale lo sviluppatore interagisce attraverso un insieme di librerie che, partendo dal basso, sono:
  - **foundation:** fornisce un'astrazione delle funzionalità di animazione, grafica e gestione<sup>[g]</sup>;
  - **rendering:** fornisce un'astrazione per gestire il layout. Con questo livello è possibile costruire un albero di oggetti renderizzabili.
  - **widget:** ogni oggetto renderizzabile ha una classe corrispondente in questa libreria, definiti appunto *widget*, l'unità fondamentale in *Flutter*, che permettono la realizzazione dell'interfaccia grafica;
  - **Material e Cupertino:** forniscono un'implementazione di alto livello dei *widget* per la realizzazione di interfacce grafiche seguendo le linee guida di *Material Design* e *Cupertino*, rispettivamente per le piattaforme *Android* e *iOS*.
- **Dart App:** codice sorgente sviluppato dall'utente, contenente i *widget* per l'implementazione della UI<sup>[g]</sup> e la logica dell'applicazione.

Inoltre, essendo *Flutter* open source<sup>[g]</sup>, il suo codice sorgente è pubblico e quindi visibile ed estendibile da qualsiasi sviluppatore.

È possibile utilizzare librerie di terze parti, sviluppate dalla *community*, per aggiungere funzionalità all'applicazione, come ad esempio librerie per la gestione dello stato dell'applicazione, per la gestione delle richieste HTTP, ecc.

Lo stato posseduto dai `StatefulWidget` (vedi Sezione 4.4) può essere considerato come *ephemeral* (ing. effimero) o *app state* (ing. stato dell'applicazione).

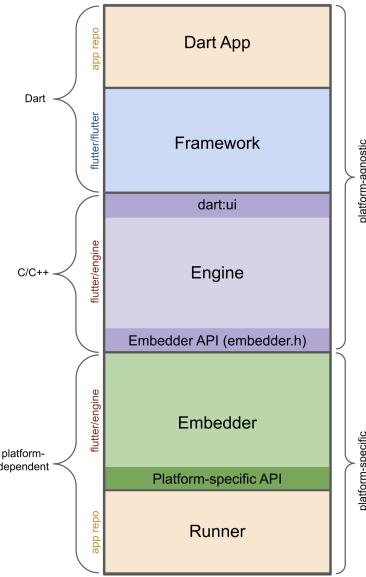
*Ephemeral* è uno stato che può essere opportunamente confinato all'interno di un singolo *widget* e gestito attraverso la primitiva `setState()`, che permette di definire come aggiornarlo.

Mentre *app state* è uno stato che viene condiviso tra più *widget* e per la sua gestione, più complessa utilizzando solamente la primitiva sopra citata, esistono diverse librerie di terze parti, ciascuna con le proprie peculiarità a seconda del caso d'uso.

La problematica di come gestire lo stato dell'applicazione è definito in gergo *state management* [29], e dopo un'attenta analisi delle librerie disponibili, si è scelto di utilizzare *Riverpod* [53].

### 4.2.2 Riverpod

*Riverpod* semplifica notevolmente lo *state management* e si basa su un concetto evoluto da *Provider* [50], libreria da cui deriva.



**Figura 4.1:** Struttura di un'applicazione [25] in *Flutter*.

A differenza di `InheritedWidget` [41], che permette di condividere lo stato tra più *widget*, `Riverpod`, che ne è una reimplementazione, fornisce *providers* che sono indipendenti dai *widget*, in quanto una volta dichiarato il `ProviderScope` a livello globale, possono essere letti ovunque, a condizione che la classe richiamante venga estesa con `ConsumerWidget` o `Consumer StatefulWidget` [51].

Da questo ne consegue che si evita di aggiornare l'interfaccia grafica, ovvero di richiamare il metodo `build()` di un *widget* quando non è necessario, poiché è un'operazione costosa.

Di *provider* ne esistono varie tipologie, per le quali si rimanda alla documentazione ufficiale [53], ma quelle implementate in questo progetto sono:

- `FutureProvider` [33]: utilizzato per ricevere un valore generato da un'operazione asincrona (es: richiesta HTTP [g]);
- `StateNotifierProvider` [63]: utilizzato per gestire una classe che mantiene uno stato, più complesso di una semplice variabile primitiva, e di esporre dei metodi per monitorarlo o aggiornarlo.

### 4.2.3 Architettura dell'applicazione

Per l'architettura dell'applicazione si è scelto di utilizzare un *pattern* architetturale basato su `MVC` [g] [24], permettendo così di separare la logica dell'applicazione dalla sua rappresentazione grafica, in modo da rendere più semplice la manutenzione e l'aggiunta di nuove funzionalità.

Nel dettaglio l'applicazione è composta da quattro livelli:

- **Data Layer**: contiene le classi che si occupano di recuperare i dati dal server, attraverso richieste HTTP [g], e di convertirli in oggetti rappresentati nel *domain layer*;

- **Domain Layer:** contiene le classi che rappresentano i dati dell'applicazione;
- **Application Layer:** contiene le classi che si occupano di gestire la logica dell'applicazione;
- **Presentation Layer:** contiene le classi che si occupano di gestire l'interfaccia, ovvero di eseguire il rendering dei *widget* e di gestire gli eventi generati dall'utente.

Questa architettura permette inoltre di avere la possibilità di definire eventualmente più sorgenti da cui recuperare i dati senza dover modificare il codice relativo ai livelli superiori, in quanto è sufficiente modificare il *data layer*.

### 4.3 Scelta della struttura del progetto

Un altro aspetto importante da considerare per la realizzazione di un progetto software è la sua struttura, ovvero come organizzare i file e le cartelle che lo compongono.

Dopo opportune ricerche ed analisi, si è scelto di adottare, tra le due alternative disponibili, la struttura *layer first*, che prevede di organizzare i file e le cartelle in base al livello a cui appartengono, in modo da rendere più semplice la manutenzione e l'aggiunta di nuove funzionalità.

*Feature first*, l'altra alternativa, organizza invece i file e le cartelle, mantenendo la separazione tra i livelli, in base alle funzionalità che l'applicazione offre.

Il motivo principale per cui la scelta non è ricaduta su quest'ultima è che, nonostante sia quella che garantisca un'organizzazione e manutenzione del codice migliore, risulta essere più complessa da implementare in quanto adatta per progetti di dimensione e complessità maggiore.

Nella Figura 4.2 verranno illustrati entrambi gli approcci, in modo da poterli confrontare e comprendere meglio le loro differenze [27].

```
// LAYER FIRST
lib/
  data/
    feature1.dart
    feature2.dart
  domain/
    feature1.dart
    feature2.dart
  application/
    feature1.dart
    feature2.dart
  presentation/
    feature1.dart
    feature2.dart

// FEATURE FIRST
lib/
  feature1/
    data.dart
    domain.dart
    application.dart
    presentation.dart
  feature2/
    data.dart
    domain.dart
    application.dart
    presentation.dart
```

**Figura 4.2:** Esempi di approccio *Layer First* (a sinistra) e *Feature First* (a destra).

### 4.4 Ambiente di sviluppo

Di seguito viene descritto l'ambiente di sviluppo e data una panoramica delle tecnologie e strumenti utilizzati.

## Figma

*Figma* [21] è un software di editor di grafica vettoriale che permette di progettare interfacce grafiche per applicazioni *web* e *mobile*.

È stato utilizzato per la realizzazione del mockup<sup>[gl]</sup> dell'applicazione, in quanto vi è la possibilità di creare un prototipo interattivo, che simula l'interazione dell'utente con l'applicazione, e di condividerlo con il team di sviluppo, in modo da avere un'idea più chiara di come l'applicazione debba essere strutturata e di come debba funzionare.

## Git

*Git* [34] è un sistema di controllo di versione, finalizzato al tracciamento del codice sorgente e delle sue modifiche, inoltre ne permette la condivisione e dunque la collaborazione tra più sviluppatori.

Inoltre è possibile, in caso di errori, ripristinare una versione precedente del codice sorgente.

## GitHub

*GitHub* [35] è un servizio di hosting per il codice sorgente di progetti software che utilizza *Git*.

Per questo progetto, è stato utilizzato per la condivisione e gestione del codice sorgente attraverso una repository dedicata, fornita dall'azienda.

Per la pianificazione della fase di implementazione, è stato utilizzato il sistema di issue tracking<sup>[gs]</sup> integrato, creando delle milestone<sup>[gl]</sup>, per ogni classe di obiettivi da raggiungere in base alla loro priorità (vedi Sezione 2.2).

In ciascuna milestone<sup>[gl]</sup>, sono state create delle *issue*, in base ai requisiti o ad un insieme di questi, necessarie per il raggiungimento di ciascun obiettivo, garantendo così una maggiore organizzazione e tracciabilità del lavoro svolto e dei progressi fatti.

## VSCode

*VSCode* è un editor di codice sorgente open source<sup>[gl]</sup>, che oltre a fornire le funzionalità di base necessarie per lo sviluppo (ad esempio: controllo di sintassi, *debugging*, analisi statica del codice, ecc.), supporta molti linguaggi di programmazione ed è possibile estendere le sue funzionalità o il numero di linguaggi supportati attraverso delle estensioni.

Di fatto per questo progetto si è reso necessario l'installazione di alcune estensioni, tra cui:

- **Flutter** [26];
- **Dart** [13].

## Flutter

*Flutter* [23] è un framework<sup>[gl]</sup> che consente di sviluppare applicazioni native per diverse piattaforme, utilizzando un unico linguaggio di programmazione, riducendo i tempi e i costi di produzione, senza compromettere le prestazioni dell'applicazione.

Il concetto centrale di *Flutter* è quello dei *widget*, oggetti che descrivono come deve essere visualizzata una parte dell'interfaccia grafica. Questi possono essere di due tipi:

- **StatelessWidget**: non hanno uno stato interno, ovvero non cambiano nel tempo, e sono definiti da un insieme di proprietà, chiamate *proprietà immutabili*, che vengono passate al costruttore del *widget*;
- **StatefulWidget**: al contrario, possiedono uno stato interno mutabile, e viene usato quando una parte dell’interfaccia utente può cambiare dinamicamente.

## StarUML

*StarUML* [62] è uno strumento che permette di modellare sistemi software, sviluppati secondo il paradigma *orientato agli oggetti*, attraverso la creazione di diagrammi UML[gl].

È stato utilizzato per la realizzazione dei casi d’uso (vedi Sezione 3.3) e dei diagrammi delle classi.

## Emulatori Android e iOS

Per effettuare i test dell’applicazione su dispositivi *Android* e *iOS*, è stato utilizzato rispettivamente *Android Studio* [2] e *Xcode* [76], che forniscono degli emulatori per le rispettive piattaforme.

## API della piattaforma RiskAPP

A completare l’ambiente di sviluppo, ci sono le API[gl] del back-end[gl], della piattaforma *RiskAPP*.

È possibile accedervi attraverso lo *Swagger* [64], strumento che, tra le altre cose, consente di consultare la documentazione delle API[gl] e di testarle attraverso un’interfaccia grafica *web*.

Si specifica che, per lo sviluppo dell’applicazione, l’utilizzo di tali API[gl] è stato svolto su un *server* di collaudo.

# Capitolo 5

# Implementazione

In questo capitolo si discuterà dell'implementazione (o codifica) dell'applicazione, in conseguenza alle scelte progettuali descritte nel capitolo precedente. Inoltre verranno descritte le librerie di terze parti utilizzate, motivandone la scelta.

## 5.1 Struttura del progetto

Seguendo quanto descritto nel capitolo 4, si è proceduto con l'implementazione del prodotto software.

Nella Figura 5.1 verrà illustrata l'effettiva struttura del progetto basata sull'approccio *layer first* (vedi Sezione 4.3), descrivendone poi, nel corso del capitolo, la varie classi contenute nei file di ciascuna cartella.

```
assets/
lib/
  components/
  constants/
  data/
    model/
    service/
  provider/
  screens
  styles/
  utils/
main.dart
```

**Figura 5.1:** Struttura del progetto basato sull'approccio *Layer First*

L'architettura citata nella Sezione 4.2.3 è stata applicata nel seguente modo: il *data layer* e *domain layer* sono stati implementati rispettivamente in `service` e `model`, contenute all'interno della cartella `data`, mentre l'*application layer* è stato implementato nella cartella `provider` e il *presentation layer* nella cartella `screens`.

Le restanti cartelle invece sono servite come ausilio per l'implementazione delle funzionalità dei *layer* sopracitati, come ad esempio `components` per la creazione di *widget* personalizzati e utilizzati dal *presentation layer*.

## 5.2 Components

In questa cartella sono raccolti tutti i *widget* personalizzati, utilizzati dalle schermate presenti in *screens* (vedi Sezione 5.6).

Di seguito verranno descritti i vari *widget* implementati, suddivisi in base alla loro funzionalità.

Si specifica inoltre che la maggior parte di essi ha una visibilità pubblica, poiché devono essere utilizzati dalle schermate. Tuttavia, alcuni hanno una visibilità privata, in quanto sono stati creati per essere utilizzati esclusivamente all'interno di altri *widget* contenuti nello stesso file.

### 5.2.1 Alert Dialogs

Nel file denominato `alert_dialogs.dart` sono stati implementati dei `StatelessWidget` che consentono di personalizzare un *alert dialog*.

Di seguito verranno elencate le classi definite in tale file, comprese di una descrizione della loro implementazione.

#### **CustomBaseAlertDialog**

È una classe che implementa un `AlertDialog` [1] e ne definisce l'aspetto base, fissando alcuni elementi: `Text` [66] per il titolo, `Widget` per l'icona posta sottostante al titolo, il suo colore e padding [g].

Mentre è possibile scegliere se aggiungere o meno un testo descrittivo e dei pulsanti di conferma e/o annulla, in base al contesto dell'operazione.

#### **IconAlertDialog**

Classe che implementa `CustomBaseAlertDialog`, personalizzandolo ulteriormente impostando con dei valori fissi sia il padding [g] che la dimensione dell'icona.

Attraverso il costruttore è obbligatorio passare il *widget* di tipo `IconData` [39], il suo colore e un titolo, mentre è opzionale passare un testo descrittivo, e se aggiungere o meno dei pulsanti di conferma e/o annulla.

#### **LoadingAlertDialog**

Personalizzazione di `IconAlertDialog`, in cui viene impostata come icona un `CircularProgressIndicator` [10] che rappresenta un indicatore di caricamento.

È possibile decidere, attraverso il costruttore, il colore dell'indicatore e il titolo da visualizzare.

Lo scopo di questa classe è quella di essere utilizzata per mostrare un indicatore di caricamento durante l'esecuzione di un'operazione asincrona.

#### **WarningAlertDialog**

Personalizzazione di `IconAlertDialog` (Figura 5.2) dove si richiede, nel costruttore, di passare un'icona e il suo colore, un titolo, il contenuto del testo descrittivo, l'azione che deve compiere il bottone di conferma alla sua pressione e il suo stile grafico (vedi Sezione 5.7.2), mentre per il pulsante di annullamento, il suo comportamento e aspetto grafico è stato fissato.

Lo scopo di questa classe è quella di essere utilizzata per mostrare un messaggio di avvertimento all'utente riguardante una scelta e la sua conferma.



**Figura 5.2:** Esempi di `WarningAlertDialog`

### ResponseDialog

Personalizzazione di `IconAlertDialog`, dove si richiede nel costruttore, di passare un’icona, il suo colore e un titolo (Figura 5.3).

Lo scopo di questa classe è quella di essere utilizzata per mostrare un messaggio di risposta all’utente riguardante l’esito di un’operazione.



**Figura 5.3:** Esempio di `ResponseDialog`

## 5.2.2 App Bars

Nel file denominato `app_bars.dart`, sono stati implementati dei `StatelessWidget` che consentono di personalizzare un *app bar*.

Di seguito verranno elencate le classi definite in tale file, comprese di una descrizione della loro implementazione.

### CustomAppBar

Classe che implementa `AppBar`, [3] definendone il colore di background, applicato dal tema dell’applicazione (vedi Sezione 5.7.4), e il titolo, in cui si tratta dell’applicazione di un’immagine vettoriale contenuta nella cartella `assets` la quale rappresenta il logo dell’azienda.

Infine è possibile scegliere se aggiungere o meno un’icona, che rappresenta il pulsante di accesso alla schermata dell’account utente (Figura 5.4).

La motivazione di quest’ultima scelta è il fatto che questa personalizzazione dell’`AppBar` viene utilizzata per la quasi totalità delle viste, anche in quella dell’account utente,

dove non si rende necessaria l'icona menzionata precedentemente poichè ci si trova già in tale vista.



**Figura 5.4:** CustomAppBar senza icona (a sinistra) e CustomAppBar con icona (a destra)

### LoginAppBar

Classe che implementa `AppBar`, personalizzandolo appositamente per la schermata di *login* (vedi Sezione 5.6.3), che è simile a quella precedente, differenziandosi per la non presenza dell'icona che funge da pulsante di accesso alla schermata dell'account utente e per l'allineamento centrale del titolo (Figura 5.5).



**Figura 5.5:** LoginAppBar

### 5.2.3 Biomteric Switch

Nel file denominato `biometric_switch.dart`, è stato implementato un `StatefulWidget` per costruire un componente in grado di permettere all'utente di abilitare il riconoscimento biometrico.

Questo *widget* è composto da un `Switch` [65] e un `Text` [66] che rappresenta il testo descrittivo (Figura 5.6).

Inoltre ne viene definito il comportamento attraverso la classe `State`, che estende `StatefulWidget`, implementando il metodo `build` per la costruzione del *widget* e il metodo `onChanged` per la gestione dell'evento di cambiamento di stato dello `Switch`.

Stato che è rappresentato da due variabili di tipo `bool`, una che si occupa di gestire l'abilitazione del componente e l'altro che indica se nel dispositivo in cui viene eseguita l'applicazione è supportato il riconoscimento biometrico.

La prima tra queste variabili, ad ogni suo cambiamento di stato, viene salvata in locale attraverso una libreria di terze parti (di cui verrà discussa nella Sezione 5.8.6) per mantenere in memoria la preferenza dell'utente.

Il caso in cui la seconda variabile, sia `false`, indica dunque l'assenza del supporto per il riconoscimento biometrico, si notifica l'utente di tale mancanza e si disabilita lo `Switch`.



**Figura 5.6:** Switch On-Off per l'abilitazione del riconoscimento biometrico

### 5.2.4 Date Picker

Nel file denominato `date_picker.dart` sono stati implementati dei *widget* per costruire dei componenti in grado di permettere all'utente di selezionare una data o un intervallo

di date.

Di seguito verranno elencate le classi definite in tale file, comprese di una descrizione della loro implementazione.

### DateButton

Questo componente estende `StatelessWidget` e si occupa di costruire un pulsante che visualizza la data, o un intervallo di date, selezionata/e dall'utente.

Ha visibilità privata, poichè è stato creato per essere utilizzato esclusivamente all'interno dei `widget` contenuti nello stesso file.

È composto da un `TextButton` [67], in cui viene definito l'aspetto grafico, e un `Text` per visualizzare la data o un intervallo di date (Figura 5.8), passato attraverso il costruttore, insieme anche al comportamento del pulsante, definito attraverso il metodo `onPressed` [45].

### CustomDateRangePicker

Questo componente estende  `StatefulWidget` e si occupa di costruire un `widget` che permette all'utente di selezionare un intervallo di date.

È composto da `DateButton`, al quale il primo parametro che viene passato è l'intervallo di date da visualizzare, di default o selezionato dall'utente, precedentemente formattato da un metodo privato `dateFormatter`: riceve in input la data iniziale e finale dell'intervallo (nel caso in cui si intenda selezionare un singolo giorno è sufficiente impostarne con esso entrambi i campi) e restituisce una stringa che rappresenta l'intervallo di date formattato opportunamente.

Mentre come secondo parametro viene passato un altro metodo privato `show` che si occupa di visualizzare il calendario e di consentire all'utente di selezionare un intervallo (Figura 5.7).

Inoltre è presente un metodo pubblico `onDateRangeSelected`, si occupa di gestire l'evento di selezione dell'intervallo di date, aggiornando lo stato del `widget`.

Lo scopo di questo `widget` è quello di fornire la possibilità all'utente di selezionare una singola data o intervallo di date per filtrare la lista di *Meeting Note* e viene impiegato solamente all'interno del `FilterPanel` (vedi Sezione 5.2.5).

### CustomDatePicker

L'implementazione di questo componente è del tutto analoga a quello citato precedentemente, `CustomDateRangePicker`, con la differenza che permette all'utente di selezionare una singola data.

Mentre il suo scopo è quello di fornire all'utente, nel momento di revisione dei dati estratti dall'elaborazione del testo da parte di un algoritmo di *intelligenza artificiale*, di modificare la data di una *Meeting Note* da creare.

## 5.2.5 Filter Panel

Nel file denominato `filter_panel.dart` è stato implementato un `Consumer StatefulWidget` [51], il cui comportamento è il medesimo di uno  `StatefulWidget`, con la differenza che è in grado di leggere i dati forniti da un *Provider* (vedi Sezione 4.2.2). Tale componente è composto da quattro `widget`, ciascuno dei quali consente di filtrare la lista di *Meeting Note* secondo i criteri definiti in fase di analisi dei requisiti, che verranno elencati di seguito (Figura 5.8):



**Figura 5.7:** Selettori di intervallo di date

- `CustomObjectPicker` (la cui implementazione è discussa nella Sezione 5.2.8);
- `CustomAutocomplete` (la cui implementazione è discussa nella Sezione 5.2.12);
- `CustomDateRangePicker` (la cui implementazione è discussa nella Sezione 5.2.4);
- `CustomToggleButtons` (la cui implementazione è discussa nella Sezione 5.2.14).

Come da prassi, ne viene definito l’aspetto grafico e il comportamento che deve avere. Per quanto riguarda quest’ultimo, che sostanzialmente consiste nel memorizzare e passare alla schermata dedicata quali filtri sono stati attivati dall’utente, tale operazione viene effettuata con l’ausilio di uno `StateNotifierProvider` (vedi Sezione 4.2.2) e una classe `FilteringOptions` (vedi Sezione 5.8.4).

### 5.2.6 Login Form

Nel file denominato `login_form.dart` è stato implementato un `ConsumerStatefulWidget` (discusso nella Sezione 4.2.2).

Lo scopo di questo componente (integrato nella schermata discussa nella Sezione 5.6.3, dove è visibile in Figura 5.20) è quello di consentire all’utente di autenticarsi all’applicazione, per farlo ci sono due modi: inserendo manualmente le credenziali oppure utilizzando il riconoscimento biometrico.

Per soddisfare il primo caso, il componente è composto da due `TextField` [70] e un `ElevatedButton` [18], di cui vengono definiti l’aspetto grafico e il comportamento.

Il `TextField` riguardante l’inserimento della password presenta anche un pulsante per mostrarla in chiaro o nasconderla.

Il pulsante di conferma è disabilitato fintanto che non vengono inserite entrambe le credenziali.



**Figura 5.8:** Popup del FilterPanel

Successivamente attraverso l'ausilio di `authProvider` (vedi Sezione 5.5.1) viene effettuata la richiesta di autenticazione, che in caso di successo porta alla schermata principale dell'applicazione, salvando il *token* di autenticazione ricevuto (vedi Sezione 5.8.6), altrimenti viene mostrato un messaggio di errore.

Mentre nel secondo caso, si effettua un controllo per verificare se l'utente in precedenza aveva abilitato tale funzionalità, in caso affermativo viene eseguito il riconoscimento biometrico (vedi Sezione 5.8.1), altrimenti si prosegue con l'inserimento manuale delle credenziali.

L'operazione e l'esito dell'autenticazione attraverso il riconoscimento biometrico, che avviene sempre attraverso `authProvider`, è simile a quello descritto per l'autenticazione manuale, con la differenza che quest'ultimo metodo preleva le credenziali dalla memoria locale, precedentemente salvate dall'operazione di abilitazione del riconoscimento biometrico da parte dell'utente (vedi Sezione 5.2.3).

Inoltre è presente una variabile di stato `isVerification` che se impostata a `true` indica che questa *form* viene impiegata come verifica delle credenziali per confermare l'abilitazione del riconoscimento biometrico (vedi Sezione 5.6.1).

### 5.2.7 Meeting Note Card

Nel file denominato `meeting_note_card.dart`, sono stati implementati dei `StatelessWidget` per costruire un componente in grado di visualizzare una *Meeting Note* in una lista, e/o in dettaglio, con la possibilità di eliminarla e/o modificarla.

Si specifica che tutti i dati necessari da visualizzare sono stati passati attraverso il costruttore.

Di seguito verranno elencate le classi definite in tale file, comprese di una descrizione della loro implementazione.

### MeetingNoteTitle

Si tratta di un *widget* che si occupa di disporre gli elementi che compongono il titolo di una *Meeting Note*, ovvero l'identificativo del cliente e la data dell'incontro, in modo tale che entrambi siano posizionati sulla stessa riga, con il primo che occupa la maggior parte dello spazio e il secondo che viene posizionato a destra (Figura 5.9).

Inoltre è stato aggiunto un *Divider* [16] che funge da separatore tra il titolo e il contenuto parziale della *Meeting Note*, visualizzabile direttamente nella lista di *Meeting Note*.

### MeetingNoteItem

*Widget* che effettivamente costruisce l'*item* della lista di *Meeting Note*, composto da un *ListTile* [42], in cui nella proprietà *title* viene posto *MeetingNoteTitle* e in *subtitle* il contenuto della *Meeting Note* stroncato ad un massimo di due righe di testo (Figura 5.19).

All'evento *onTap* del componente, viene mostrato a schermo una modale che appare dal basso, contenente i dettagli della *Meeting Note*.

Si vuole precisare che, contrariamente a quanto pensato durante la realizzazione del mockup<sup>[g]</sup>, si è deciso di apportare una modifica per quanto riguarda la visualizzazione dei dettagli della *Meeting Note*, in quanto essendo l'applicazione usata da utenti in mobilità, è più ergonomico mostrare i dettagli in una modale che appare dal basso, piuttosto che dall'espansione di un *item*.

### MeetingNoteCard

Componente contenuto nella modale menzionata precedentemente, il quale si occupa di disporre tutti gli elementi di cui è composta una *Meeting Note* (Figura 5.9).

È composto da un *MeetingNoteTitle*, il contenuto della *Meeting Note*, l'autore e i due pulsanti che permettono di eliminarla e/o modificarla, il comportamento della prima azione viene passata per parametro, mentre per la seconda viene definita direttamente in quanto è stato sufficiente rimandare alla schermata del *wizard* per poi effettuare le modifiche (vedi Sezione 5.6.5 a 5.6.7).

#### 5.2.8 Object Picker

Nel file denominato *object\_picker.dart* è stato implementato *CustomObjectPicker*, classe che estende un *StatefulWidget* per costruire un componente in grado di permettere all'utente di selezionare la categoria dei clienti.

Viene utilizzato all'interno del *Filter Panel* (vedi Sezione 5.2.5), dal *wizard* di creazione/modifica di una *Meeting Note* (vedi Sezione 5.6.5) e nella schermata di revisione per la creazione automatica di una *Meeting Note* (vedi Sezione 5.6.4).

Anche per questo componente si è pensato di rivisitarlo rispetto al mockup<sup>[g]</sup>, in quanto si è deciso di utilizzare un *CupertinoPicker* [12], che visualizza le categorie dei clienti selezionabili, passate in input attraverso il costruttore, al posto di un *DropdownMenu* [17], questo per favorire l'utilizzo dell'applicazione da parte di utenti in mobilità, poiché il primo appare dal basso e dunque diventa più facilmente raggiungibile (Figura 5.10). Inoltre sono stati definiti dei metodi che permettono di ottenere la categoria selezionata attraverso l'indice e di aggiornare lo stato del *widget* al cambiamento di quest'ultima.



**Figura 5.9:** Meeting Note Card



**Figura 5.10:** Selettori di categoria dei clienti

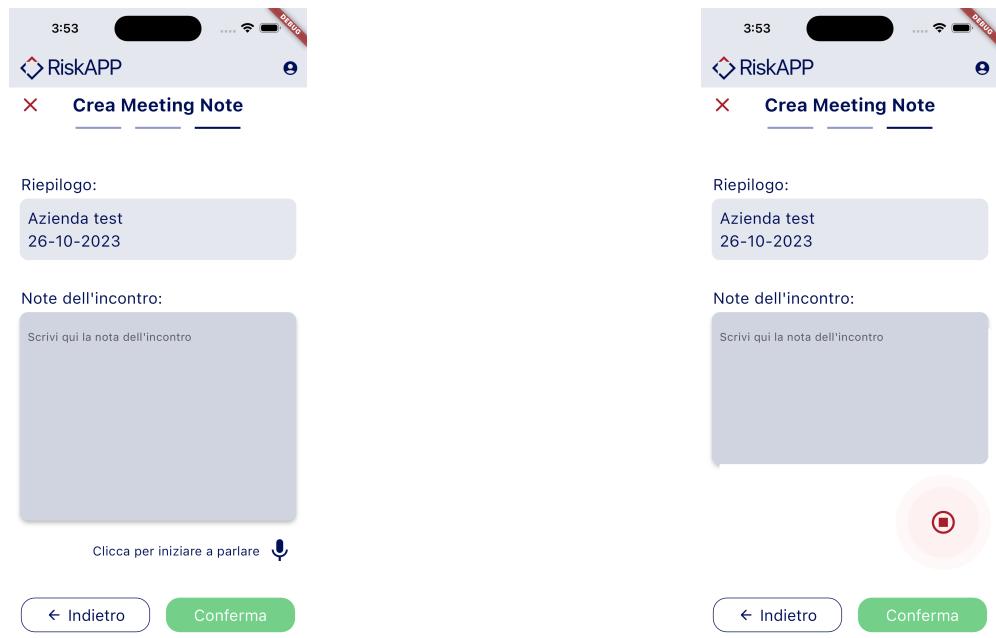
### 5.2.9 Recording Button

Nel file denominato `recording_button.dart`, è stato implementato un `StatelessWidget` per costruire un componente in grado di permettere all'utente di attivare la dettatura

vocale.

È composto da un **Text** [66] che esplicita all'utente lo scopo del pulsante, da un **IconButton** [38] e da un **AvatarGlow** [5] per rappresentare l'animazione che viene visualizzata quando la dettatura vocale è attiva.

L'attivazione avviene in base al valore della variabile **bool isRecording**, passata attraverso il costruttore, come anche il comportamento del pulsante, definito attraverso il metodo **onPressed** [45] (vedi Figura 5.11).



**Figura 5.11:** Pulsante nello stato di non attivazione (a sinistra) e attivazione (a destra) della dettatura vocale

### 5.2.10 Screens Template

Nel file denominato `screens_template.dart`, sono stati implementati dei `StatelessWidget` per costruire diversi componenti che rappresentano gli elementi comuni per la struttura delle schermate dell'applicazione.

Di seguito verranno elencate le classi definite in tale file, comprese di una descrizione della loro implementazione.

#### BaseScreen

Classe in cui viene definito la struttura e l'aspetto base che dovranno avere tutte le schermate dell'applicazione.

È composto da un `Scaffold` [54], contenitore principale di tutti gli elementi grafici, in cui viene applicato il tema dell'applicazione (vedi Sezione 5.7.4).

Sono state definite poi varie sue proprietà di base, tra cui `appBar` con `CustomAppBar` (vedi Sezione 5.2.2) e un `body`, passato per parametro del costruttore, in quanto ogni schermata ha il suo contenuto.

È stata inoltre definita la proprietà `bottomNavigationBar`, che non viene utilizzata da tutte le schermate, ma vi è comunque la possibilità di implementarla nel caso cui si vogliono aggiungere dei pulsanti di navigazione nella parte inferiore della vista per avanzare o retrocedere tra le varie schermate dell'applicazione.

Per abilitare tali pulsanti è sufficiente passare, a seconda delle necessità, un `widget forwardButton` per la progressione e/o `backButton` per la retrocessione.

#### WizardScreen

Classe che implementa `BaseScreen` e che si occupa di fornire un *template* per le schermate che compongono il wizard [g] di creazione/modifica di una *Meeting Note* (descritte nella Sezione 5.6).

Imposta a `true` la proprietà `enableIcon`, mantenendo il pulsante di accesso alla schermata dell'account utente.

Attraverso i parametri del costruttore è obbligatorio passare il `body`, mentre sono opzionali `forwardButton` e `backButton`.

#### WizardHeader

Classe che definisce l'aspetto grafico dell'*header* del wizard [g] di creazione/modifica di una *Meeting Note*.

È composto da un `Text` [66] che rappresenta il titolo della schermata e da un `IconButton` [38], quest'ultimo ha lo scopo di mostrare un `AlertDialog` (vedi Sezione 5.2.1), in cui chiede all'utente se vuole abbandonare il wizard [g], in caso affermativo si viene riportati alla schermata principale dell'applicazione, altrimenti viene chiusa la modale (Figura 5.12).



**Figura 5.12:** WizardHeader

### WizardStepper

Classe che è composta da tre `Step` (vedi Sezione 5.2.10), il numero massimo di passi pensato per il wizard<sup>[g]</sup>: riceve in input il numero di passo corrente in modo da evidenziare a che punto l'utente si trova nella progressione. Si specifica che `WizardStepper` è privato, è stato infatti creato con lo scopo di essere utilizzato esclusivamente all'interno di `WizardHeader` (Figura 5.13).



**Figura 5.13:** *WizardStepper*

### Step

Classe che definisce l'aspetto grafico di un passo del wizard<sup>[g]</sup> di creazione/modifica di una *Meeting Note*.

Sostanzialmente consiste in un `Divider` [16] che rappresenta un singolo passo, sono stati poi definiti due colori diversi per evidenziare quello in cui l'utente si trova da quelli precedenti e/o successivi.

### 5.2.11 Scroll Date Picker

Nel file denominato `scroll_date_picker.dart`, è stata definita la classe `CustomScrollDatePicker` che estende `StatefulWidget` per costruire un componente in grado di permettere all'utente di selezionare una data, nel momento di creazione/modifica di una *Meeting Note* (Sezione 5.6.6).

È composto da un `Text` [66] che visualizza la data selezionata, un `DatePicker` [56] e un pulsante che reimposta la data selezionata a quella corrente.

Espone dei metodi che permettono di ottenere la data selezionata con `selectedDate`, e di aggiornare lo stato del `widget`, al cambiamento di quest'ultima con `onDateChanged`.

### 5.2.12 Search Bar

Nel file denominato `search_bar.dart` sono stati implementati dei `widget` che estendono  `StatelessWidget` per costruire dei componenti in grado di permettere all'utente di effettuare una ricerca.

Di seguito verranno elencate le classi definite in tale file, comprese di una descrizione della loro implementazione.

### CustomearchBar

Classe che implementa una `SearchBar` [58] in cui viene definito pressoché solo l'aspetto grafico in quanto attraverso il costruttore è obbligatorio passare un `TextEditingController` [68], si occupa di gestire il testo inserito dall'utente, e un metodo `onChanged` che gestisce l'evento di cambiamento del testo inserito dall'utente, mentre è opzionale passare il colore del `background` del componente.

Lo scopo è quello di fornire all'utente la possibilità di effettuare una ricerca all'interno della lista dei clienti (questo componente, essendo integrato nella schermata discussa nella Sezione 5.6.5, è visibile in Figura 5.23).

### CustomAutocomplete

Classe che implementa `Autocomplete` [4] (Figura 5.14), consente di fornire dei suggerimenti per l'autocompletamento nella ricerca, in cui viene anche qui definito solo l'aspetto grafico.

Attraverso il costruttore si rende necessario passare dei metodi fondamentali: `optionsBuilder` fornisce i suggerimenti per l'autocompletamento, `displayStringForOption` definisce quale attributo dell'oggetto suggerito visualizzare, mentre `onSelected` si occupa di gestire l'evento di selezione di un suggerimento.

Viene impiegato nel `widget FilterPanel` (vedi Sezione 5.2.5) per fornire all'utente la possibilità di ricercare un cliente e nella schermata di conferma per la creazione automatica di una *Meeting Note* (vedi Sezione 5.6.4)



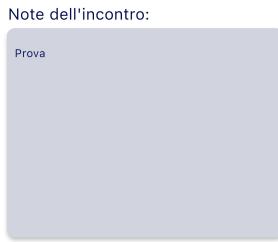
**Figura 5.14:** *Autocomplete*

#### 5.2.13 Text Box

Nel file denominato `text_box.dart` è stato implementato un `StatefulWidget` per costruire un componente in grado di permettere all'utente di inserire del testo.

È composto da semplicemente un `TextField` [69] in cui viene personalizzato nell'aspetto grafico, è richiesto dal costruttore il passaggio dell'altezza di questo componente, in modo da poterlo utilizzare in diverse situazioni, poi è necessario passare un `TextEditingController` [68] e un metodo `onChanged` che si occupa di gestire l'evento di cambiamento del testo inserito dall'utente.

Viene utilizzato per la creazione/modifica di una *Meeting Note* e per la creazione automatica.



**Figura 5.15:** *TextBox*

### 5.2.14 Toogle Buttons

Nel file denominato `toogle_buttons.dart`, è stato implementato un `StatefulWidget` per costruire un componente in grado di permettere all'utente di selezionare una delle due opzioni disponibili (Figura 5.16).

È composto da un `ToggleButtons` [74], di cui viene definito l'aspetto grafico e il comportamento.

Attraverso il costruttore, è necessario passare un `List<Widget>` contenente le due opzioni disponibili, un `List<bool>` che indica quale opzione è stata selezionata e un metodo `onChoiceSelected` che si occupa di gestire l'evento di selezione di un'opzione.

Viene impiegato nel `FilterPanel` (vedi Sezione 5.2.5) per fornire all'utente la possibilità di selezionare una delle due opzioni disponibili per filtrare la lista di *Meeting Note*.



**Figura 5.16:** *ToggleButtons*

### 5.2.15 Warning Alert

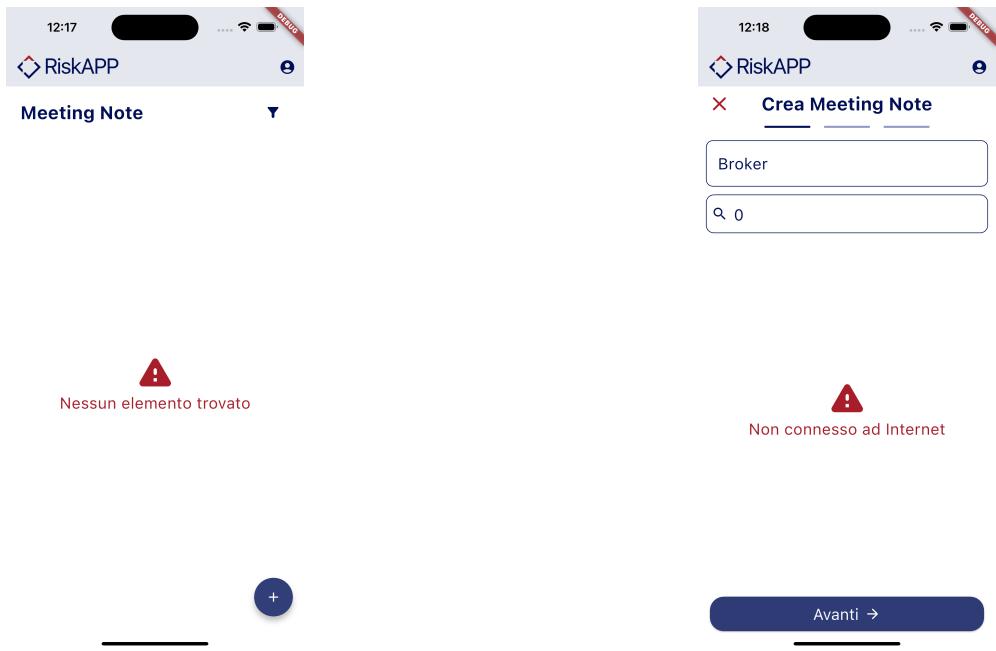
Nel file denominato `warning_alert.dart`, è stato implementato un  `StatelessWidget` per costruire un componente composto da un `Text` [66] e un `Icon` [37], il cui scopo è quello di visualizzare un messaggio di avvertimento, attraverso i parametri del costruttore viene passato tale messaggio e il colore che devono avere il testo e l'icona (Figura 5.17). I casi in cui viene richiesto si faccia riferimento ai seguenti requisiti: RFN-12, RFN-13, RFN-20, RFN-30, RFN-34, RFN-64.

## 5.3 Constants

In questa cartella sono presenti file che contengono delle costanti utilizzate all'interno dell'applicazione.

Le costanti sono incapsulate all'interno di classi, raggruppate in base al loro utilizzo, e sono state definite in maniera statica così da essere richiamate senza dover istanziare un oggetto.

Lo scopo è quello di centralizzare tutte le costanti, in modo da poterle modificare facilmente e da poterle utilizzare in maniera uniforme all'interno dell'applicazione.



**Figura 5.17:** Esempi di *WarningAlert*

### 5.3.1 App constants

Nel file denominato `app_constants.dart` sono state definite delle costanti di tipo `String` che rappresentano ad esempio i nomi delle schermate, dei campi di una *Meeting Note* o il contenuto testuale di alcuni pulsanti, ecc.

### 5.3.2 API constants

Nel file denominato `api_constants.dart` sono state definite due classi che contengono le costanti utilizzate per effettuare le richieste al back-end<sup>[g]</sup>.

Nella prima classe `ApiUrls`, sono presenti le costanti che si riferiscono agli endpoint<sup>[g]</sup> dell'API<sup>[g]</sup>, mentre nella seconda, `ApiFields`, sono presenti quelle che si riferiscono al nome dei parametri che devono essere passati all'URI<sup>[g]</sup> di alcune richieste al back-end<sup>[g]</sup>.

L'utilizzo di queste costanti avviene esclusivamente all'interno delle classi contenute nella cartella `services` (vedi Sezione 5.4.2).

## 5.4 Data

Questa cartella contiene due cartelle, ciascuna rappresentante il *data layer* e *domain layer* dell'applicazione, le quali verranno ampiamente discusse di seguito.

### 5.4.1 Model

In questa cartella sono contenuti i file che rappresentano il modello dei dati utilizzato all'interno dell'applicazione.

Si specifica, per evitare ridondanze, che alcune classi in questione sono state implementate seguendo un *pattern* specifico: oltre alla definizione degli attributi opportuni, ovvero quelli specificati nelle API<sup>[g]</sup> del back-end<sup>[g]</sup>, sono presenti un costruttore *factory* [20] `fromJson`, come si intuisce si occupa di convertire da JSON<sup>[g]</sup> all'oggetto interessato, dei metodi che ritornano il valore di ciascun attributo e un metodo `toString` che ritorna una stringa rappresentante un'istanza della classe.

Quest'ultimo metodo è stato definito con il solo scopo di visualizzare il contenuto di un oggetto in fase di *debugging*. Nel corso di questa Sezione verrà specificato quando una classe è stata implementata seguendo tale *pattern*, mentre per le altre verranno descritte nel dettaglio.

Inoltre in quasi tutti i file è presente una classe, il cui nome termina con suffisso `Response`, che estende `BaseResponse` (vedi Sezione 5.4.1) e rappresenta l'esito di una richiesta HTTP<sup>[g]</sup>fatta al back-end<sup>[g]</sup>, nel caso in cui l'esito sia positivo contiene anche il valore da restituire.

Infine, per ogni classe, è sottintesa la presenza del costruttore con tutti gli attributi presenti.

Di seguito verranno elencati i file, al cui interno sono definite delle classi che costituiscono il modello dei dati.

### Authentication

In questo file sono state definite tutte le classi necessarie per eseguire l'operazione di autenticazione all'applicazione, di seguito verrà descritta l'implementazione di ciascuna.

#### AuthArgs

Classe che incapsula i parametri necessari per effettuare la richiesta di autenticazione, è composta da due attributi di tipo `String` che rappresentano lo *username* e la *password* dell'utente.

Inoltre espone due metodi che si occupano di effettuare degli overriding<sup>[g]</sup> dei metodi `operator ==` [47] e `hashCode` [36], in modo da effettuare un confronto tra due oggetti di tipo `AuthArgs`.

Infine è presente un metodo `toString` che ritorna una stringa rappresentante un'istanza della classe.

#### AuthValues

Classe che incapsula il valore restituito dal back-end<sup>[g]</sup> in seguito alla richiesta di autenticazione, è composta da un attributo di tipo `String` che rappresenta il *token* di autenticazione.

La sua implementazione è stata effettuata seguendo il *pattern* definito nell'introduzione di questa Sezione.

#### AuthResponse

Classe che estende `BaseResponse` (vedi Sezione 5.4.1), nella quale viene aggiunto un attributo di tipo `AuthValues` che ne rappresenta il valore restituito.

### Meeting Note

In questo file sono state definite tutte le classi necessarie per gestire i dati riguardanti le *Meeting Note* all'interno dell'applicazione, di seguito verrà descritta l'implementazione di ciascuna.

#### RawMeetingNote

Classe che si occupa di incapsulare i dati di una *Meeting Note*, ricevuti da una chiamata al back-end<sup>[g]</sup>, che però non sono ancora utilizzabili all'interno dell'applicazione, la quasi totalità degli attributi di tipo `String` ed alcuni di essi dunque non sono del tipo opportuno per la loro elaborazione all'interno dell'applicazione.

Gli attributi in questione sono:

- `uuid`, utilizzato per l'identificazione di una *Meeting Note*;
- `meetingDate`, la data dell'incontro;
- `note`, il contenuto della *Meeting Note*;
- `userCreator`, l'autore della *Meeting Note*.

Inoltre sono presenti tre attributi opzionali di tipo `MeetingNoteObject`, ciascuno rappresentante un cliente<sup>[g]</sup> delle tre categorie: tra i quali solo uno di loro viene restituito da una chiamata al *backend* e dunque non è nullo, poiché una *Meeting Note* può essere associata ad un solo cliente<sup>[g]</sup>.

Una considerazione necessaria riguarda l'attributo `meetingDate`, che non è di tipo `DateTime` [15], dunque non è possibile effettuare ad esempio un confronto tra due date, situazione analoga per l'attributo `userCreator` che è di tipo `String` e non `User`.

Per questo motivo è stata definita la classe `MeetingNote` (discussa di seguito) che si occupa di effettuare la conversione da `RawMeetingNote` a `MeetingNote` e permette quindi di operare più facilmente su questi dati.

La sua implementazione è stata effettuata seguendo il *pattern* definito nell'introduzione di questa Sezione, con l'aggiunta di un ulteriore metodo `toJSON`, che si occupa di convertire l'oggetto in JSON<sup>[g]</sup> per poterlo inviare al back-end<sup>[g]</sup>.

#### PaginatedMeetingNote

Classe che si occupa di incapsulare una lista di `RawMeetingNote` paginata, con l'ulteriore presenza di due attributi di tipo `String` che rappresentano rispettivamente il *link* alla pagina precedente e successiva.

La sua implementazione è stata effettuata seguendo il *pattern* definito nell'introduzione di questa Sezione.

#### MeetingNote

Classe che si occupa di incapsulare i dati di una *Meeting Note*, pronta per essere utilizzata all'interno dell'applicazione, è composta dai seguenti attributi:

- `uuid` di tipo `String`, utilizzato per l'identificazione di una *Meeting Note*;
- `meetingDate` di tipo `DateTime` [15], la data dell'incontro;

- `note` di tipo `String`, il contenuto della *Meeting Note*;
- `user` di tipo `User`, l'autore della *Meeting Note*;
- `object` di tipo `MeetingNoteObject`, il cliente<sup>[g]</sup> associato alla *Meeting Note*.

La sua implementazione, molto semplice, consiste nei metodi per ottenere i valori degli attributi e un metodo `toString` che ritorna una stringa che rappresenta l'oggetto.

### **MeetingNoteResponse**

Classe che estende `BaseResponse`, nella quale viene aggiunto un attributo di tipo `List<MeetingNote>` che ne rappresenta il valore restituito.

### **SmartCreationValue**

Classe che si occupa di incapsulare i dati, risultati dell'elaborazione da parte di un algoritmo di IA<sup>[g]</sup> per la creazione automatica di una *Meeting Note*, è composta dai seguenti attributi:

- `customer` di tipo `String`, il nome del cliente<sup>[g]</sup> elaborato dall'algoritmo;
- `date` di tipo `DateTime`, la data dell'incontro elaborato dall'algoritmo;
- `category` di tipo `String`, la categoria del cliente<sup>[g]</sup> elaborato dall'algoritmo;
- `note` di tipo `String`, il contenuto della *Meeting Note* elaborato dall'algoritmo.

La sua implementazione è stata effettuata seguendo il *pattern* definito nell'introduzione di questa Sezione, con un ulteriore metodo `toJson` che si occupa di convertire l'oggetto in JSON<sup>[g]</sup> per poterlo inviare al back-end<sup>[g]</sup> e un altro metodo `getObjectCategory` che converte l'attributo `category` in un oggetto di tipo `ObjectCategory`.

### **SmartCreationResponse**

Classe che estende `BaseResponse`, nella quale viene aggiunto un attributo di tipo `SmartCreationValue` che ne rappresenta il valore restituito.

### **Meeting Note Object**

In questo file sono state definite tutte le classi necessarie per gestire i dati riguardanti i clienti all'interno dell'applicazione, di seguito verrà descritta l'implementazione di ciascuna.

### **MeetingNoteObject**

Classe che rappresenta il modello di un cliente relativo ad una *Meeting Note*, ed è composta dai seguenti attributi:

- `uuid`, di tipo `String`, utilizzato per l'identificazione di un cliente<sup>[g]</sup>;
- `name`, di tipo `String`, il nome del cliente<sup>[g]</sup>;

- `vatNumber`, di tipo `String`, il numero di partita IVA del cliente ed è presente solo per i *contraenti*;
- `category`, di tipo `ObjectCategory`, la categoria del cliente<sup>[g]</sup>.

La sua implementazione è stata effettuata seguendo il *pattern* definito nell'introduzione di questa Sezione.

### PaginatedObject

Dalle specifiche dell'API<sup>[g]</sup>, solamente i clienti di tipo *broker* e *contraente* vengono restituiti in modo paginato, per questo motivo è stata definita questa classe, che rappresenta un *wrapper* di una lista di `MeetingNoteObject` e di due altri attributi di tipo `String`, che rappresentano rispettivamente il *link* alla pagina precedente e successiva.

La sua implementazione è stata effettuata seguendo il *pattern* definito nell'introduzione di questa Sezione.

### ObjectValue

È una classe *sealed* [57] in cui è stato definito un costruttore di *default* e un metodo `toString` che ritorna una stringa rappresentante l'oggetto.

### ObjectListValue

Classe che estende `ObjectValue` e rappresenta il valore restituito in seguito alla richiesta di ottenere la lista dei clienti, indipendentemente a quale categoria appartengono e se sono paginati o meno.

Ridefinisce il costruttore e il metodo `toString`, mentre viene aggiunto un metodo per ottenere la lista dei clienti.

### SingleObjectValue

Classe che estende `ObjectValue` e rappresenta il valore restituito in seguito alla richiesta di ottenere un singolo cliente, la sua implementazione è analoga a quella di `ObjectListValue`.

### MeetingNoteObjectResponse

Classe che estende `BaseResponse`, nella quale viene aggiunto un attributo di tipo `ObjectValue` che ne rappresenta il valore restituito.

### Object Category

Si tratta semplicemente di una enumerazione che rappresenta le tre categorie dei clienti<sup>[g]</sup>:

- `ObjectCategory.broker`: rappresenta la categoria dei broker;
- `ObjectCategory.contractor`: rappresenta la categoria dei contraenti;

- `ObjectCategory.delegatedInsurer`: rappresenta la categoria delle compagnie assicurative.

### Status Response

La classe astratta `BaseResponse` modella l'esito di una qualsiasi richiesta HTTP<sup>[g]</sup>fatta al back-end<sup>[g]</sup>.

Ha un attributo di tipo `StatusResponse`, una enumerazione che rappresenta tre possibili stati di una richiesta HTTP<sup>[g]</sup>:

- `StatusResponse.success`: la richiesta è andata a buon fine;
- `StatusResponse.unauthorizedException`: la richiesta non è andata a buon fine a causa del *token* di autenticazione scaduto (vedi Sezione 5.8.3);
- `StatusResponse.genericException`: la richiesta non è andata a buon fine per un errore generico.

Inoltre espone due metodi, il primo si occupa di restituire il valore dell'attributo `StatusResponse` e il secondo di restituire una stringa che rappresentante un'istanza della classe.

### User

In questo file sono state definite tutte le classi necessari per gestire l'utente all'interno dell'applicazione, di seguito verrà descritta l'implementazione di ciascuna.

#### User

Classe che modella l'utente all'interno dell'applicazione, gli attributi che la compongono sono: `email`, `firstName`, `lastName`, di tipo `String`, e `profile` di tipo `Profile`.

La sua implementazione è stata effettuata seguendo il *pattern* definito nell'introduzione di questa Sezione.

#### Profile

Classe che modella il profilo dell'utente all'interno dell'applicazione, gli attributi che la compongono sono: `uuid`, un identificativo univoco assegnato ad ogni utente, e l'`avatar`, contenente il link dell'immagine profilo, entrambi di tipo `String`.

La sua implementazione è stata effettuata seguendo il *pattern* definito nell'introduzione di questa Sezione.

#### UserResponse

Classe che estende `BaseResponse`, nella quale viene aggiunto un attributo di tipo `User` che ne rappresenta il valore restituito.

### 5.4.2 Services

In questa Sezione verranno discusse le classi che si occupano di effettuare le richieste HTTP<sup>[g]</sup>al back-end<sup>[g]</sup>attraverso le API<sup>[g]</sup>, per evitare ridondanze, ogni servizio, eccetto `BaseService`, gestisce le eventuali eccezioni in caso una chiamata non vada a buon fine.

### Authentication Service

In questo file è definita una classe che estende `BaseService` (classe discussa di seguito), con la composizione di `PostRequest` definisce il metodo `getToken`, che si occupa di effettuare una richiesta POST al back-end<sup>[g]</sup>.

Riceve in input un oggetto di tipo `AuthArgs` (vedi Sezione 5.4.1) e restituisce un oggetto di tipo `AuthResponse`.

### Base Service

In questo file sono state definite tutte le classi necessarie per fornire un’astrazione per le chiamate HTTP<sup>[g]</sup>, di seguito verrà descritta l’implementazione di ciascuna.

### BaseService

Classe astratta che si occupa di fornire un’interfaccia per gestire le chiamate HTTP<sup>[g]</sup> e provvede a definire i metodi fondamentali per effettuare una richiesta:

- `getUrl`, che si occupa di restituire l’URI<sup>[g]</sup> della risorsa data una stringa in input;
- `handleResponse`, invocata per ogni tipologia di richiesta, che si occupa di determinare se essa è avvenuta con successo o meno, in caso di quest’ultimo viene lanciata un’eccezione.

Le seguenti classi, che sono di tipo `mixin` [44], aggiungono funzionalità alla classe `BaseService`: ogni classe `mixin` si occupa di fornire un’implementazione per una chiamata HTTP<sup>[g]</sup> specifica.

Successivamente, quando verranno utilizzate, la classe in questione dovrà estendere `BaseService` e includere alcune di queste classi `mixin`, in modo da aggiungere ad un servizio le chiamate necessarie.

Di seguito verranno descritte tali classi, per ciascuna si specificheranno i parametri necessari per la configurazione dell’`header` o del `body` della richiesta, a seconda delle necessità.

### GetRequest

Classe `mixin` che si occupa di fornire un’implementazione per effettuare una richiesta GET, i parametri necessari richiesti nell’`header` sono l’`url` della risorsa e il `token` di autenticazione.

### PatchRequest

Classe `mixin` che si occupa di fornire un’implementazione per effettuare una richiesta PATCH, i parametri necessari sono l’`url` della risorsa, il `token` di autenticazione e il contenuto per il `body` della richiesta.

### DeleteRequest

Classe `mixin` che si occupa di fornire un’implementazione per effettuare una richiesta DELETE, i parametri necessari sono l’`url` della risorsa e il `token` di autenticazione.

### PostRequest

Classe `mixin` che si occupa di fornire un'implementazione per effettuare una richiesta `POST`.

Vi sono presenti due metodi, il primo si occupa di effettuare una `POST` con lo scopo di salvare dei dati sul *backend*, i parametri necessari sono l'`url` della risorsa, il `token` di autenticazione e il `body` della richiesta.

Il secondo si occupa di effettuare una `POST` con lo scopo di ottenere il `token` di autenticazione, l'unico parametro necessario è di tipo `AuthArgs`.

### Meeting Note Object Service

In questo file è definita una classe che estende `BaseService` (classe discussa in precedenza) e che, con la composizione di `GetRequest`, definisce i seguenti metodi:

- `getMeetingNoteObject`, ritorna `MeetingNoteObjectResponse` (vedi Sezione 5.4.1) in seguito alla richiesta di ottenere un singolo cliente<sup>[g]</sup> per la creazione di una *Meeting Note*, per fare ciò è necessario passare il `token` di autenticazione, l'`url` della risorsa, includendo il parametro `uuid` del cliente e la sua categoria;
- `getContractors`, ritorna `MeetingNoteObjectResponse` (vedi Sezione 5.4.1) in seguito alla richiesta di ottenere la lista paginata dei clienti<sup>[g]</sup> di tipo *contraente*, per fare ciò è necessario passare obbligatoriamente il `token` di autenticazione, `pageSize` e `pageNumber`, rappresentanti rispettivamente la dimensione della pagina e il numero della pagina, opzionalmente è possibile poi passare anche il nome del cliente da ricercare;
- `getBrokers`, implementazione analoga a `getContractors` ma per i clienti<sup>[g]</sup> di tipo *broker*;
- `getDelegatedInsurers`, ritorna `MeetingNoteObjectResponse` (vedi Sezione 5.4.1) in seguito alla richiesta di ottenere la lista non paginata dei clienti<sup>[g]</sup> di una *compagnia assicurativa*, per fare ciò è necessario passare il `token` di autenticazione ed opzionalmente il nome del cliente da ricercare;
- `buildUrl`, si occupa di costruire l'URI<sup>[g]</sup> della risorsa in base ai parametri passati, che sono: l'`url` della risorsa, la dimensione della pagina, il numero della pagina e il nome del cliente da ricercare, esso viene utilizzato dai metodi `getContractors` e `getBrokers`.

### Meeting Note Service

In questo file è definita una classe che estende `BaseService` (classe discussa in precedenza) e che, con la composizione di `GetRequest`, `PostRequest`, `PatchRequest` e `DeleteRequest`, definisce i seguenti metodi:

- `getMeetingNotes`, ritorna `MeetingNoteResponse` (vedi Sezione 5.4.1) in seguito alla richiesta di ottenere la lista paginata delle *Meeting Note*, per fare ciò è necessario passare obbligatoriamente il `token` di autenticazione, `pageSize` e `pageNumber`, rappresentanti rispettivamente la dimensione della pagina e il numero della pagina, opzionalmente è possibile passare anche un oggetto `FilteringOptions` (vedi Sezione 5.8.4) che rappresenta le opzioni di filtraggio.  
Ad ogni chiamata riceve la lista di `RawMeetingNote` e la converte in una lista di `MeetingNote` (vedi Sezione 5.4.1);

- `postMeetingNote`, ritorna `StatusResponse` in seguito alla richiesta di creare una *Meeting Note*, per fare ciò è necessario passare il *token* di autenticazione e un oggetto di tipo `MeetingNote` (vedi Sezione 5.4.1), convertendo quest'ultimo in JSON<sup>[g]</sup> attraverso un metodo dedicato;
- `patchMeetingNote`, ritorna `StatusResponse` in seguito alla richiesta di modificare una *Meeting Note*, per fare ciò è necessario passare il *token* di autenticazione e un oggetto di tipo `MeetingNote` (vedi Sezione 5.4.1), convertendo quest'ultimo in JSON<sup>[g]</sup> attraverso un metodo dedicato;
- `deleteMeetingNote`, ritorna `StatusResponse` in seguito alla richiesta di eliminare una *Meeting Note*, per fare ciò è necessario passare il *token* di autenticazione e l'*uuid* della *Meeting Note*;
- `postSmartCreationMeetingNote`, ritorna `SmartCreationResponse` (vedi Sezione 5.4.1) in seguito alla richiesta di creare una *Meeting Note* in modo automatico, per fare ciò è necessario passare il *token* di autenticazione e il testo da elaborare;
- `buildUrl`, si occupa di costruire l'URI<sup>[g]</sup> della risorsa in base ai parametri passati, che sono: la dimensione della pagina, il numero della pagina ed eventualmente le opzioni di filtraggio, esso viene utilizzato dal metodo `getMeetingNotes`.

### User Service

In questo file è definita una classe che estende `BaseService` (classe discussa in precedenza) e che, con la composizione di `GetRequest`, definisce il metodo `getUser` che si occupa di effettuare una richiesta GET al back-end<sup>[g]</sup> per ottenere i dati dell'utente, ricevendo in input il *token* di autenticazione e tornando un oggetto di tipo `UserResponse`.

## 5.5 Provider

In questa cartella sono presenti le classi che si occupano di gestire la *logica di business* dell'applicazione, si tratta dunque dell'*application layer* (vedi Sezione 4.2.3).

In ciascun file viene definita una classe, al cui interno sono stati implementati dei metodi opportuni per effettuare delle operazioni e al di fuori di essa, si istanzia un `FutureProvider` [33] (discusso nella Sezione 4.2.2) per esporre questi metodi al *presentation layer*.

Inoltre, ciascuna classe per effettuare le operazioni necessarie si avvale dei corrispettivi servizi (vedi Sezione 5.4.2) e modelli (vedi Sezione 5.4.1) precedentemente discussi.

Si vuole infine precisare che, eccetto per `AuthenticationHelper` (vedi Sezione 5.5.1), nelle restanti classi, ogni metodo che effettua una richiesta include l'operazione di lettura del *token* per autenticare quest'ultime.

### 5.5.1 Authentication Provider

In questo file è definita la classe `AuthenticationHelper` che si occupa di gestire la logica dell'autenticazione all'applicazione, in particolare espone i seguenti metodi:

- `verifyCredentials`, si occupa di verificare la validità delle credenziali, passate per parametro e inserite dall'utente;

- **authenticate**, verifica le credenziali dell’utente, avvalendosi del metodo `verifyCredentials` e, in caso di successo, ritorna un *token* di autenticazione attraverso l’oggetto `AuthResponse` salvandolo poi in locale (vedi Sezione 5.8.6);
- **logout**, si occupa di effettuare il *logout* dall’applicazione, eliminando il *token* di autenticazione salvato in locale e riportando l’utente alla schermata di *login*;
- **isAuthenticated**, verifica se l’utente è autenticato, controllando se il *token* di autenticazione è salvato in locale;
- **isUnauthorized**, verifica se l’utente è autorizzato ad effettuare una chiamata HTTP<sup>[g]</sup>, controllandone l’esito e in caso negativo effettua il *logout* dall’applicazione, poichè significa che il *token* di autenticazione è scaduto;
- **isBiometricEnabled**, verifica se l’utente ha abilitato l’autenticazione biometrica all’interno dell’applicazione, controllando se il *flag* è salvato in locale;
- **readCredentials**, si occupa di leggere le credenziali cifrate salvate in locale (vedi Sezione 5.8.6), restituendo un oggetto di tipo `AuthArgs` (vedi Sezione 5.4.1);
- **saveCredentials**, si occupa di salvare le credenziali cifrate in locale, ricevendo in input un oggetto di tipo `AuthArgs` (vedi Sezione 5.4.1);
- **removeCredentials**, si occupa di eliminare le credenziali cifrate salvate in locale.

### 5.5.2 Meeting Note Object Provider

In questo file è definita la classe `MeetingNoteObjectHelper`, si occupa di gestire la logica dei clienti<sup>[g]</sup> all’interno dell’applicazione e in particolare espone i seguenti metodi:

- **getContractors** si occupa di ottenere la lista paginata di tutti i clienti<sup>[g]</sup> di tipo *contraente* oppure solo di quelli ricercati dall’utente attraverso il passaggio del parametro `searchTerm`, dati in input il numero della pagina e la dimensione della pagina, restituendo un oggetto di tipo `MeetingNoteObjectResponse`;
- **getBrokers** implementazione analoga a `getContractors` ma per i clienti<sup>[g]</sup> di tipo *broker*;
- **getDelegatedInsurers** implementazione analoga ai due metodi precedenti ma per i clienti<sup>[g]</sup> di tipo *compagnia assicurativa* e senza la possibilità di paginazione.

### 5.5.3 Meeting Note Provider

In questo file è definita la classe `MeetingNoteHelper`, si occupa di gestire la logica delle *Meeting Note* all’interno dell’applicazione e in particolare espone i seguenti metodi:

- **fetchMeetingNotes**, si occupa di ottenere la lista paginata delle *Meeting Note* oppure solo di quelle filtrate dall’utente attraverso il passaggio del parametro `filteringOptions` (vedi Sezione 5.8.4), dati in input il numero della pagina e la dimensione della pagina, restituendo un oggetto di tipo `MeetingNoteResponse`;
- **createMeetingNote**, si occupa di memorizzare la *Meeting Note* creata e passata in input come oggetto di tipo `MeetingNote` nel back-end<sup>[g]</sup>, restituendo un oggetto di tipo `StatusResponse` in quanto è sufficiente ritornare l’esito della richiesta;

- `deleteMeetingNote`, si occupa di eliminare la *Meeting Note* passata in input lo `uuid` come parametro, restituendo un oggetto di tipo `StatusResponse` in quanto è sufficiente ritornare l'esito della richiesta;
- `modifyMeetingNote`, si occupa di modificare la *Meeting Note* passata in input come oggetto di tipo `MeetingNote`, restituendo un oggetto di tipo `StatusResponse` in quanto è sufficiente ritornare l'esito della richiesta;
- `smartCreation`, si occupa della creazione di *Meeting Note* in modo automatico, passando in input il testo da elaborare, restituendo un oggetto di tipo `SmartCreationResponse`.

#### 5.5.4 User Provider

In questo file è definita la classe `UserHelper` che si occupa di gestire la logica dell'utente all'interno dell'applicazione, in particolare espone il metodo `initUser` che si occupa di ottenere i dati dell'utente, restituendo un oggetto di tipo `UserResponse`.

### 5.6 Screens

In questa cartella sono presenti tutte le schermate dell'applicazione, le quali verranno descritte di seguito.

#### 5.6.1 Account Screen

`AccountScreen` è la schermata che permette all'utente di visualizzare i propri dati, letti attraverso `UserProvider` (vedi Sezione 5.5.4), la cui classe estende `ConsumerWidget` (vedi Sezione 4.2.2) in modo da essere in grado di invocare un *provider*.

Per la definizione dell'aspetto grafico, questa schermata implementa `BaseScreen` (vedi Sezione 5.2.10) e, prima di renderizzare ulteriori componenti, effettua un controllo sullo stato della connessione ad internet, ottenuto attraverso il `NetworkAwareProvider` (vedi Sezione 5.8.5), in caso di assenza di connessione viene mostrato un `WarningAlert` (vedi Sezione 5.2.15) con il relativo messaggio di errore, altrimenti viene mostrato il contenuto della schermata, servendosi di `FutureBuilder` [32] per invocare il metodo `initUser` e ottenere i dati dell'utente.

In particolare vi possono essere vari esiti:

- *waiting*, viene mostrato un `CircularProgressIndicator` [10] per indicare che la richiesta è in corso;
- *unauthorizedException*, l'utente non è autorizzato ad effettuare la richiesta, viene reindirizzato alla schermata di *login*;
- *genericException*, la richiesta non è andata a buon fine per un errore generico e viene mostrato un `WarningAlert` con il relativo messaggio di errore;
- *success*, la richiesta è andata a buon fine e viene mostrato il contenuto della schermata.

In quest'ultimo caso, vengono renderizzati i seguenti componenti (Figura 5.18):

- `TextButton` [67], effettua il *logout* dall'applicazione, alla sua pressione mostra un `WarningAlertDialog` (vedi Sezione 5.2.15) informando l'utente dell'evento, che ne deve confermare o meno l'operazione;

- **CircleAvatar** [9], mostra l'immagine profilo dell'utente;
- **AccountCard**, *StatelessWidget* privato, creato appositamente con lo scopo di definire l'aspetto grafico per la visualizzazione del nome, cognome e la mail dell'utente;
- **BiometricOption**, *StatefulWidget* privato che implementa **BiometricSwitch** (vedi Sezione 5.2.3), si ricorda che lo stato di questo componente è salvato in locale (vedi Sezione 5.8.6); per la sua attivazione verrà richiesto all'utente di confermare l'operazione attraverso l'inserimento delle credenziali di autenticazione, vengono poi cifrate e salvate in locale (vedi Sezione 5.8.6).



**Figura 5.18:** Account Screen

### 5.6.2 Home Screen

**HomeScreen** è la schermata principale dell'applicazione, la cui classe estende **Consumer StatefulWidget** (vedi Sezione 4.2.2) in modo da essere in grado di invocare un *provider*.

Prima della renderizzazione dell'aspetto grafico, viene effettuata l'inizializzazione delle variabili di stato, in particolare:

- **filteringOptions**, oggetto di tipo **FilteringOptions** (vedi Sezione 5.8.4) che incapsula le opzioni di filtraggio attive;
- **isFiltering**, **bool** che indica se l'utente ha attivato o meno almeno un filtro;
- **pagingController**, oggetto di tipo **PagingController** [40] che si occupa di gestire la paginazione dei dati.

Per la definizione dell'aspetto grafico, questa schermata implementa **BaseScreen** (vedi Sezione 5.2.10) il cui contenuto è strutturato come segue (Figura 5.19):

- **HomeHeader**,  *StatelessWidget* privato che mostra il titolo della schermata e due pulsanti, uno per eliminare i filtri attivati e l'altro per aprire il **FilterPanel** (vedi Sezione 5.2.5);
- **SpeedDial**, componente che mostra un **FloatingActionButton** [22] che consente all'utente di scegliere la modalità di creazione di una *Meeting Note*, in base alla scelta quest'ultimo verrà reindirizzato alla schermata corrispondente;
- **PagedListView** [40], componente che si occupa di mostrare la lista paginata delle *Meeting Note*, in particolare si avvale del **PagingController** per gestire la paginazione dei dati, lista che è composta da **MeetingNoteItem** (vedi Sezione 5.2.7) che definisce la rappresentazione grafica della *Meeting Note* e permette all'utente di modificarla o eliminarla.

Nel dettaglio, prima della visualizzazione della lista paginata, si effettua un controllo dello stato della connessione ad internet attraverso il **NetworkAwareProvider** (vedi Sezione 5.8.5), in caso di assenza di connessione viene mostrato un **WarningAlert** (vedi Sezione 5.2.15) con il relativo messaggio di errore.

Altrimenti viene mostrato il contenuto della schermata servendosi del metodo privato **fetchPage**, che si occupa di effettuare la richiesta al back-end<sup>[g]</sup> attraverso **MeetingNoteProvider** (vedi Sezione 5.5.3), controllando se l'utente è autorizzato ad effettuare la richiesta, attraverso il metodo **isUnauthorized** fornito da **authProvider** (vedi Sezione 5.5.1), per poi restituire la lista paginata delle *Meeting Note* e aggiornare lo stato dell'oggetto **PagingController**.

È presente un ulteriore metodo privato, **showWarningDialog**, che si occupa di mostrare un **WarningAlertDialog** (vedi Sezione 5.2.15), chiedendo conferma all'utente, per la eliminazione di una *Meeting Note* e in base alla scelta dell'utente e all'esito della richiesta, vengono gestiti i medesimi casi illustrati nella Sezione 5.4.1, paragrafo *StatusResponse*.

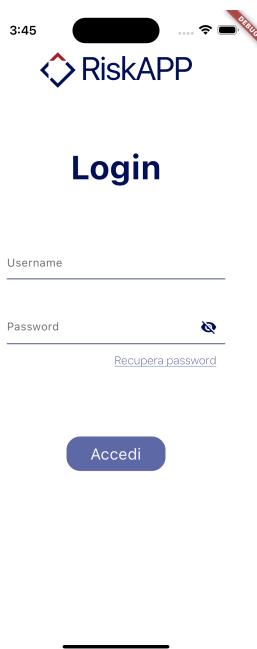


**Figura 5.19:** Home Screen

### 5.6.3 Login Screen

`LoginScreen` è la schermata che permette all'utente di effettuare l'autenticazione all'applicazione.

Definisce semplicemente l'aspetto grafico della schermata (Figura 5.20): implementa `BaseScreen` (vedi Sezione 5.2.10), impostando `LoginAppBar` (vedi Sezione 5.2.2) e nel `body` viene definito il titolo e richiamato il `LoginForm` (vedi Sezione 5.2.6), che si occupa di gestire il comportamento e per questo motivo tale classe viene estesa con `StatelessWidget`.



**Figura 5.20:** Login Screen

### 5.6.4 Smart Creation Screen

La schermata, che permette all'utente di creare una *Meeting Note* in modo automatico, è stata suddivisa in tre classi, data la sua complessità, in cui di seguito verranno descritte nel dettaglio.

#### SmartCreationPage

`SmartCreationPage` è la classe che si occupa di ricevere in input il contenuto testuale dall'utente che verrà elaborato da un algoritmo di *intelligenza artificiale*, implementato dal tutor aziendale, per la creazione automatica di *Meeting Note*.

Prima della renderizzazione dell'aspetto grafico viene effettuata l'inizializzazione delle variabili di stato, in particolare:

- `textController`, `TextEditingController` [68] che si occupa di gestire il contenuto testuale inserito dall'utente;

- `isButtonEnabled`, `bool` che indica se l'utente ha inserito del testo e abilita di conseguenza il pulsante che ne avvia l'elaborazione;
- `speechToText`, `SpeechToText` [61] che si occupa di gestire la dettatura vocale;
- `listenedWords`, `String` che memorizza quanto detto dall'utente.

Per la definizione dell'aspetto grafico, questa schermata implementa `WizardScreen` (vedi Sezione 5.2.10) e il suo contenuto è strutturato come segue (Figura 5.21):

- `WizardHeader`, imposta il titolo della schermata (vedi Sezione 5.2.10);
- `CustomTextBox`, componente che si occupa di ricevere in input il contenuto testuale inserito dall'utente, che può avvenire in due modalità differenti come specificato nei requisiti RFN-72 e RFN-73 (vedi Sezione 5.2.13);
- `RecordingButton`, componente che si occupa di attivare la dettatura vocale (vedi Sezione 5.2.9);
- `ElevatedButton` [18], componente che si occupa di avviare l'elaborazione del testo inserito dall'utente.

Nel dettaglio, quest'ultima operazione viene effettuata servendosi di `meetingNoteProvider` per effettuare l'elaborazione e di `authProvider` per autorizzare la richiesta attraverso il metodo `isUnauthorized`, inoltre viene mostrato un `CircularProgressIndicator` [10] per indicare che la richiesta è in corso.

Al termine dell'elaborazione, con i tre possibili esiti (i medesimi illustrati nella Sezione 5.4.1) e, in caso di successo viene mostrato `ConfirmCreationPopup` attraverso una modale che appare dal basso.



**Figura 5.21:** Smart Creation Screen

### ConfirmCreationPopup

`ConfirmCreationPopup` è la classe che si occupa di mostrare all'utente i dati estratti dall'elaborazione del testo, passati in input attraverso il costruttore, dall'oggetto di tipo `SmartCreationValue`. Inoltre permette, all'utente, di confermare la creazione della *Meeting Note* o di eventualmente apportare delle modifiche ad alcuni campi.

Prima della renderizzazione dell'aspetto grafico viene effettuata l'inizializzazione delle variabili di stato, in particolare:

- `selectedCategory`, `ObjectCategory` che rappresenta la categoria del cliente [g]estratto dall'elaborazione del testo;
- `selectedCategoryIndex`, `int` che rappresenta l'indice della categoria del cliente [g];
- `initialValue`, `String` che viene inizializzato con il nome del cliente [g]estratto dall'elaborazione del testo;
- `debouncedSearch` (vedi Sezione 5.8.2), si occupa di gestire la ricerca del cliente [g]attraverso il componente `CustomAutocomplete`;
- `selectedMeetingNoteObject`, `MeetingNoteObject` che si occupa di gestire eventuali modifiche sulla ricerca dell'identificativo del cliente apportate dall'utente;
- `isButtonEnabled`, `bool` che abilita il pulsante di creazione se tutti i campi sono correttamente compilati;
- `isNameFound`, `bool` che indica se l'identificativo del cliente [g], estratto dall'algoritmo sia stato trovato o meno;
- `selectedDate`, `DateTime` che rappresenta la data estratta dall'elaborazione del testo;
- `textController`, `TextEditingController` che si occupa di gestire il contenuto testuale inserito dall'utente.;

La struttura di questa modale è la seguente (Figura 5.22):

- `TitleLabel`, mostra il titolo della *label* per ogni campo elencato di seguito;
- `CustomObjectPicker`, componente che si occupa di mostrare la lista delle categorie dei clienti [g]e di permettere eventualmente all'utente di apportare modifiche, il cui funzionamento è supportato dalle variabili di stato `selectedCategory` e `selectedCategoryIndex` (vedi Sezione 5.2.8);
- `CustomAutoComplete`, componente che visualizza l'identificativo del cliente e permette all'utente di apportare modifiche effettuando una ricerca, il suo funzionamento è supportato dalle variabili di stato `initialValue`, `debouncedSearch` e `selectedMeetingNoteObject`, imposta inoltre il valore di `isNameFound` in base all'esito della ricerca (vedi Sezione 5.2.12);
- `CustomDatePicker`, componente che visualizza la data estratta dall'elaborazione del testo e permette all'utente di apportare modifiche, il suo funzionamento è supportato dalla variabile di stato `selectedDate` (vedi Sezione 5.2.4);

- `CustomTextBox`, componente che visualizza il contenuto dell'incontro estratto dall'elaborazione del testo e permette all'utente di apportare modifiche, il suo funzionamento è supportato dalla variabile di stato `textController` (vedi Sezione 5.2.13);
- `OutlineButton` [48], componente che si occupa di confermare la creazione della *Meeting Note* richiamando il metodo `onConfirmation`.

Vi sono presenti inoltre due metodi privati, `search` e `onConfirmation`.

Il primo si occupa di effettuare la ricerca dell'identificativo del cliente<sup>[g]</sup>, ricevuto input come parametro, con l'ausilio di `meetingNoteObjectProvider` (vedi Sezione 5.5.2) e di controllare l'autorizzazione della richiesta con `authProvider`, prima di tutto ciò viene assicurata la presenza di connessione ad internet da `networkAwareProvider` (vedi Sezione 5.8.5).

Vengono gestiti inoltre i vari esiti della richiesta (i medesimi illustrati nella vedi Sezione 5.4.1, paragrafo *Status Response*) e solamente in caso di successo viene restituito il risultato e aggiornato lo stato di `selectedMeetingNoteObject`.

Mentre il secondo mostra all'utente una finestra di dialogo attraverso la quale si chiede all'utente di confermare la creazione della *Meeting Note* e in caso affermativo viene effettuata la richiesta al back-end<sup>[g]</sup> attraverso `meetingNoteProvider` (vedi Sezione 5.5.3), seguendo poi lo stesso algoritmo descritto sopra.

### TitleLabel

`TitleLabel` è la classe privata che estende `StatelessWidget` e definisce solamente l'aspetto grafico del titolo per la label di ogni campo della *Meeting Note*.



**Figura 5.22:** Confirm Creation Popup

### 5.6.5 Wizard Screen 1

La schermata inherente al primo *step* per la creazione/modifica di una *Meeting Note*, permette all'utente di selezionare il cliente<sup>[g]</sup>e data la sua complessità, è stata suddivisa in classi, che verranno descritte nel dettaglio di seguito.

#### WizardPage1

Per comprendere in quale modalità viene richiamato il wizard<sup>[g]</sup>, la classe `WizardPage1` prevede un metodo privato `isEditingMode`: restituisce `true` se l'utente sta modificando una *Meeting Note* e `false` se sta creando una nuova *Meeting Note*.

Questa verifica avviene controllando l'oggetto `MeetingNote` passato in input alla classe `WizardPage1` se è vuoto, in tal caso l'utente sta creando una nuova *Meeting Note*, altrimenti sta modificando una *Meeting Note* esistente.

Prima della renderizzazione dell'aspetto grafico viene effettuata l'inizializzazione delle variabili di stato, in particolare:

- `selectedCategory`, `ObjectCategory` che assume il valore della categoria del cliente della *Meeting Note*, passata in input se l'utente sta modificando, altrimenti viene inizializzato di `default` con `ObjectCategory.broker`;
- `selectedCategoryIndex`, `int` che assume il valore dell'indice della categoria del cliente della *Meeting Note*, passata in input se l'utente sta modificando, altrimenti viene inizializzato di `default` con 0;
- `searchController`, `TextEditingController` che assume il valore dell'identificativo del cliente della *Meeting Note*, passata in input se l'utente sta modificando, altrimenti viene inizializzato di `default` con una stringa vuota;
- `selectedMeetingNoteObject`, `MeetingNoteObject` che assume il valore del cliente della *Meeting Note*, passata in input se l'utente sta modificando, altrimenti viene inizializzato di `default` con `null`;
- `pagingController`, `PagingController` che si occupa di gestire la paginazione dei dati;
- `searchTerm`, `String` che assume il valore dell'identificativo del cliente della *Meeting Note*, passata in input se l'utente sta modificando con il conseguente aggiornamento di `pagingController`, altrimenti viene inizializzato di `default` con una stringa vuota;
- `isButtonEnabled`, `bool` che abilita il pulsante di creazione se tutti i campi sono correttamente compilati e viene inizializzato a `true` se l'utente sta modificando.

Per la definizione dell'aspetto grafico, questa schermata implementa `WizardScreen` (vedi Sezione 5.2.10) e il suo contenuto è strutturato come segue (Figura 5.23):

- `WizardHeader`, imposta il titolo della schermata;
- `CustomObjectPicker`, componente che si occupa di mostrare la lista delle categorie dei clienti<sup>[g]</sup>e di permettere all'utente di effettuare una selezione, aggiornando di conseguenza `pagingController` e filtrando la lista dei clienti con la categoria selezionata, il suo funzionamento è supportato dalle variabili di stato `selectedCategory` e `selectedCategoryIndex` (vedi Sezione 5.2.8);

- `CustomSearchBar`, componente che visualizza l'identificativo del cliente e permette all'utente di effettuare una ricerca, aggiornando eventualmente `pagingController` e filtrando opportunamente la lista dei clienti, il suo funzionamento è supportato dalle variabili di stato `searchController` e `searchTerm` (vedi Sezione 5.2.12);
- `MeetingNoteObjectList`, componente che si occupa di mostrare la lista dei clienti opportunamente filtrata e paginata, il suo funzionamento è supportato dalle variabili di stato `selectedMeetingNoteObject`, il cui scopo è di evidenziare quale cliente è stato selezionato e `pagingController`;
- `ElevatedButton [18]`, componente che si occupa di confermare la selezione del cliente e di mostrare la schermata successiva del wizard<sup>[g]</sup>.

Prima della visualizzazione della lista paginata, si effettua un controllo dello stato della connessione ad internet attraverso il `NetworkAwareProvider` (vedi Sezione 5.8.5), in caso di assenza di connessione viene mostrato un `WarningAlert` (vedi Sezione 5.2.15) con il relativo messaggio di errore.

Sono presenti dei metodi privati a supporto del funzionamento del wizard<sup>[g]</sup>, che sono:

- `fetchPage`, si occupa di effettuare la richiesta al back-end<sup>[g]</sup>, attraverso il `MeetingNoteObjectProvider` (vedi Sezione 5.5.2), controllando se l'utente è autorizzato ad effettuare la richiesta attraverso il metodo `isUnauthorized`, fornito da `authProvider` (vedi Sezione 5.5.1), per poi restituire la lista dei clienti e aggiornare lo stato del `PagingController`;
- `isEditMode`, discusso all'inizio di questa Sezione;
- `showWizardPage2`, si occupa di mostrare la schermata successiva del wizard<sup>[g]</sup>, passando in input l'oggetto `MeetingNote`, solamente in caso di modifica, e l'oggetto `MeetingNoteObject` selezionato dall'utente.

### **MeetingNoteObjectList**

`MeetingNoteObjectList` è la classe che si occupa di mostrare la lista dei clienti, opportunamente filtrata e paginata.

Attraverso il costruttore, riceve in input: l'oggetto `PagingController`, la funzione `onObjectSelected`, che si occupa di aggiornare lo stato di `selectedMeetingNoteObject` e la variabile `isEditMode` per evidenziare il cliente selezionato dall'utente.

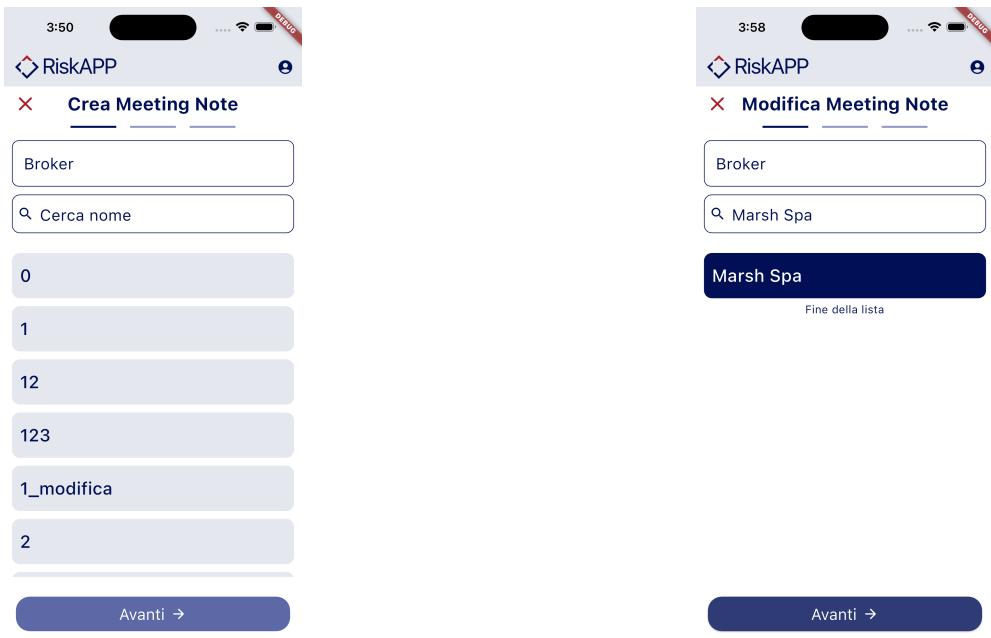
Prima della renderizzazione dell'aspetto grafico, viene effettuata l'inizializzazione della variabile di stato `selectedIndex`, che assume il valore 0 (si precisa che il conteggio degli elementi della lista parte da 0 e non da 1) se l'utente sta modificando, in quanto la lista in questa modalità mostra solamente un elemento, altrimenti viene inizializzato a -1 per indicare che nessun elemento è stato ancora selezionato.

Per la definizione dell'aspetto grafico, questa schermata implementa `StatefulWidget` e il suo contenuto è strutturato sostanzialmente da `PagedListView [40]`, componente che si occupa di mostrare la lista paginata dei clienti, in particolare si avvale del `PagingController` per gestire la paginazione dei dati, inoltre per ogni elemento della lista viene mostrato un `MeetingNoteObjectItem` (vedi Sezione 5.6.5) che rappresenta il cliente e permette all'utente di selezionarlo.

### **MeetingNoteObjectItem**

`MeetingNoteObjectItem` è la classe che si occupa di visualizzare il cliente in un *item* della lista e permette all'utente di selezionarlo.

Attraverso il costruttore riceve in input i seguenti parametri:



**Figura 5.23:** Wizard Screen 1 - Creazione (a sinistra) e Modifica (a destra)

- `isContractor, bool` che indica se il cliente è un cliente<sup>gli</sup>appartiene alla categoria dei *contraenti*;
- `vatNumber, String` che rappresenta la partita iva del contraente;
- `name, String` che rappresenta l'identificativo del cliente;
- `index, int` che rappresenta l'indice del cliente nella lista;
- `selectedIndex, int` che rappresenta l'indice del cliente selezionato dall'utente;
- `onTap, Function` che si occupa di impostare `selectedIndex` con l'indice del cliente selezionato dall'utente.

Per la definizione dell'aspetto grafico, questa schermata implementa `StatefulWidget` e il suo contenuto è strutturato sostanzialmente da `ListTile` [42], `widget` che si occupa di mostrare il cliente nella lista, in particolare si avvale dei parametri `title` e `subtitle` per mostrare rispettivamente l'identificativo del cliente e la sua partita iva, inoltre evidenzia l'*item* del cliente selezionato variandone il colore di sfondo e del testo.

Per capire quale cliente è stato selezionato dall'utente, viene effettuato una comparazione del valore di `selectedIndex` e `index`.

### 5.6.6 Wizard Screen 2

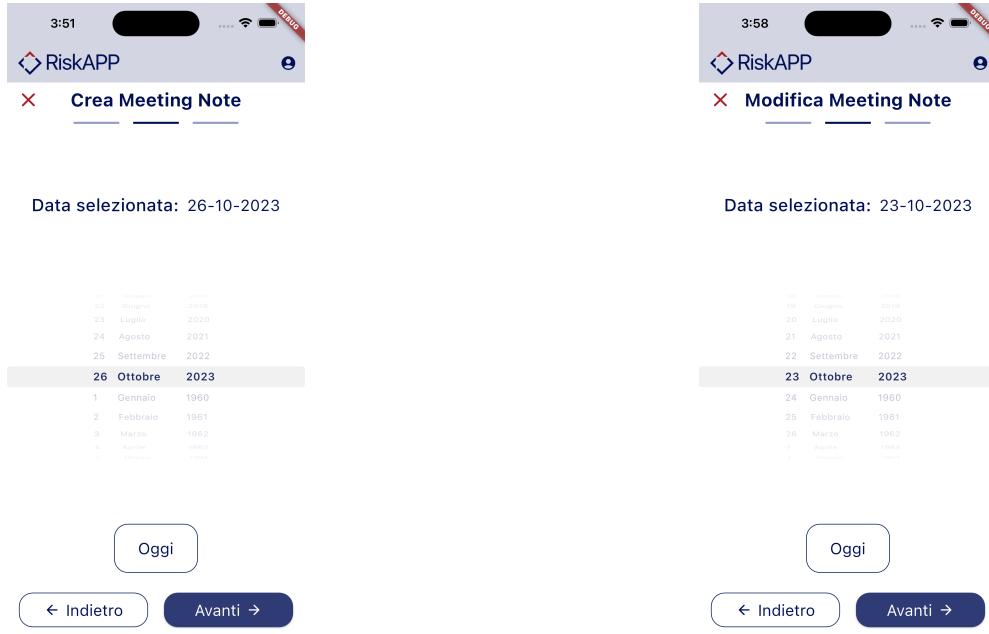
La schermata inerente al secondo *step* per la creazione/modifica di una *Meeting Note*, permette all'utente di selezionare la data in cui è avvenuto un incontro con un cliente. Prima della renderizzazione del suo aspetto grafico, viene effettuata l'inizializzazione della variabile di stato `selectedDate` che assume il valore della data della *Meeting Note* passata in input, se l'utente sta modificando, altrimenti viene inizializzato di *default*

con la data corrente (Figura 5.24).

Per la definizione dell'aspetto grafico questa schermata implementa `WizardScreen` (vedi Sezione 5.2.10) e il suo contenuto è strutturato come segue:

- `WizardHeader`, imposta il titolo della schermata (vedi Sezione 5.2.10);
- `CustomScrollDatePicker`, componente che si occupa di mostrare la data e permette all'utente di selezionarla, il suo funzionamento è supportato dalla variabile di stato `selectedDate` (vedi Sezione 5.2.11);
- `ElevatedButton [18]`, pulsante che si occupa di confermare la selezione della data e di reindirizzare alla schermata successiva del wizard<sup>[g]</sup> invocando il metodo `showWizardPage3`;
- `OutlinedButton [48]`, pulsante che si occupa di reindirizzare alla schermata precedente del wizard<sup>[g]</sup>.

In questa schermata non è presente alcun `Provider`, in quanto non è necessario effettuare alcuna richiesta al back-end<sup>[g]</sup>, poiché si tratta solamente di effettuare una selezione della data.



**Figura 5.24:** Wizard Screen 2 - Creazione (a sinistra) e Modifica (a destra)

### 5.6.7 Wizard Screen 3

La schermata inerente al terzo *step* per la creazione/modifica di una *Meeting Note*, si occupa di ricevere in input il contenuto testuale dall'utente, di conseguenza permette a quest'ultimo di confermare la creazione della *Meeting Note*.

Prima della renderizzazione dell'aspetto grafico viene effettuata l'inizializzazione delle variabili di stato, in particolare:

- `textController`, `TextEditingController` che si occupa di gestire il contenuto testuale inserito dall'utente, se sta modificando la *Meeting Note*, viene inizializzato con il suo contenuto, altrimenti con una stringa vuota;
- `isButtonEnabled`, `bool` che indica se l'utente ha inserito del testo e abilita di conseguenza il pulsante per confermare la creazione della *Meeting Note*;
- `speechToText`, `SpeechToText` che si occupa di gestire la dettatura vocale;
- `listenedWords`, `String` che memorizza quanto detto dall'utente.

Per la definizione dell'aspetto grafico, questa schermata implementa `WizardScreen` (vedi Sezione 5.2.10) e il suo contenuto è strutturato come segue (Figura 5.25):

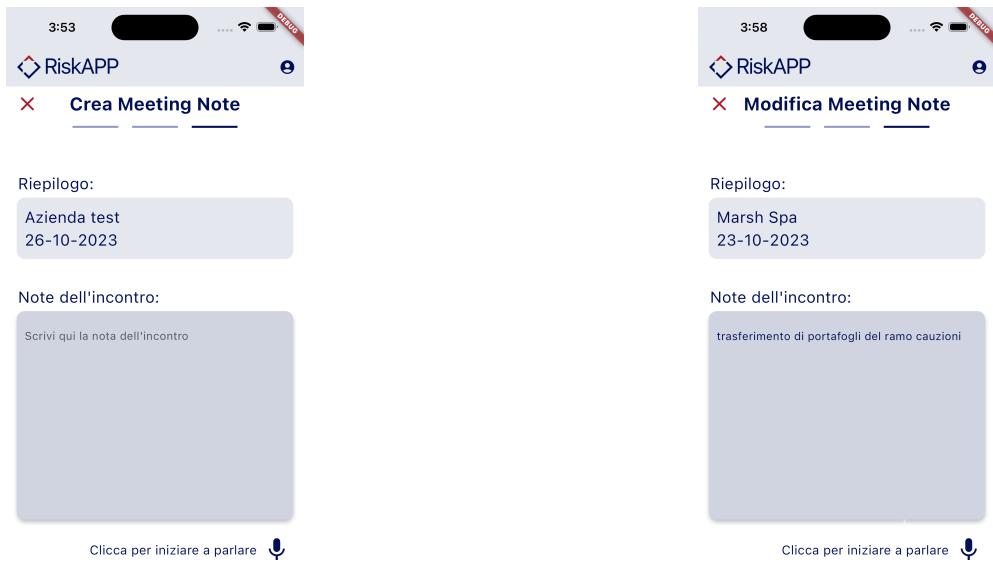
- `WizardHeader`, imposta il titolo della schermata (vedi Sezione 5.2.10);
- `SummaryCard`, classe privata, sviluppata con lo scopo di visualizzare un riepilogo dei dati inseriti dall'utente, il suo funzionamento è supportato dalle variabili di stato `meetingNoteObject` e `meetingNoteDate`, la sua struttura, verrà discussa nel dettaglio in seguito;
- `CustomTextBox`, componente che si occupa di ricevere in input il contenuto testuale inserito dall'utente, può avvenire in due modalità differenti, come specificato nei requisiti RFN-72 e RFN-73 (vedi Sezione 5.2.13);
- `RecordingButton`, componente che si occupa attivare la dettatura vocale (vedi Sezione 5.2.9);
- `OutlinedButton` [48], pulsante che si occupa di reindirizzare alla schermata precedente del wizard [g];
- `ElevatedButton` [18], pulsante che si occupa di confermare la creazione/modifica della *Meeting Note* richiamando il metodo `onConfirmation`.

Il metodo privato `onConfirmation` si occupa di mostrare all'utente una finestra di dialogo, la quale chiede all'utente di confermare la creazione della *Meeting Note* e in caso affermativo viene effettuata la richiesta al back-end [g] attraverso `meetingNoteProvider` (vedi Sezione 5.5.3), controllandone l'autorizzazione con `authProvider` e gestendo i vari esiti della richiesta (i medesimi illustrati nella Sezione 5.4.1, paragrafo *Status Response*).

Per differenziare il caso di creazione a quello di modifica si utilizza lo stesso approccio descritto nella Sezione 5.6.5, ovvero si verifica se l'oggetto `MeetingNote` passato in input è vuoto, in tal caso l'utente sta creando una nuova *Meeting Note*, altrimenti sta modificando una *Meeting Note* esistente, di conseguenza viene invocato la chiamata opportuna.

Prima della gestione degli esiti possibili della richiesta, viene effettuato un controllo sullo stato della connessione ad internet attraverso il `NetworkAwareProvider` (vedi Sezione 5.8.5), in caso di assenza di connessione viene mostrato un `WarningAlert` (vedi Sezione 5.2.15) con il relativo messaggio di errore.

Infine, durante l'elaborazione della richiesta, viene mostrata una finestra di dialogo che indica all'utente che l'operazione è in corso.



**Figura 5.25:** Wizard Screen 3 - Creazione (a sinistra) e Modifica (a destra)

## 5.7 Styles

Per applicare uno stile grafico all’interfaccia, *Flutter* [23] si serve della classe `Theme` [72] in cui viene richiamata nel `main.dart` (vedi Sezione 5.9), la conseguenza è che i *widget* implementati erediteranno lo stile grafico definito.

Senza una ridefinizione del tema, verrà applicato quello di *default*, per questo progetto invece, seguendo il mockup<sup>[8]</sup>, è stato personalizzato.

In particolare, è sufficiente ridefinire opportunamente la classe `ThemeData` [73], che contiene tutta la configurazione grafica dell’applicazione.

Per la ridefinizione del tema, si sono create delle classi, raccolte in opportuni file, che si occupano di definire i colori, dimensioni e stili grafici utilizzati nell’applicazione.

Si specifica che le variabili utilizzate per la configurazione sono definite come `static const`, in quanto devono essere accessibili senza la necessità di istanziare la classe.

Di seguito verrà illustrato nel dettaglio il contenuto di ogni file.

### 5.7.1 AppColors

`AppColors` è una classe in cui viene ridefinita la *palette* colori utilizzata nell’applicazione, in particolare:

- `primary`, rappresenta il colore primario, applicato ai componenti principali dell’applicazione (es. pulsanti, icone, ecc.);
- `secondary`, rappresenta il colore secondario dell’applicazione, applicato ai componenti secondari (es. *filter chips* [8]);
- `error`, rappresenta il colore da applicare nelle eccezioni (es. validazione dei campi);

- `surface`, rappresenta il colore di sfondo per una categoria di componenti (es. `card` [7]);
- `background`, rappresenta il colore di sfondo per l'intera applicazione.

### 5.7.2 ButtonStyles

Nel file `button_styles.dart` sono presenti differenti classi, il cui scopo è di ridefinire lo stile grafico dei pulsanti utilizzati nell'applicazione.

Di seguito verrà discusso in dettaglio la configurazione dello stile grafico dei pulsanti di tipo `ElevatedButton` [18], si precisa dunque, che per le altre due tipologie di pulsanti, `OutlinedButton` [48] e `TextButton` [67], è stato seguito lo stesso approccio.

#### ButtonsColor

`ButtonsColor` è una classe in cui viene ridefinito il colore dei pulsanti, in particolare:

- `disabledButton`, colore applicato ai pulsanti disabilitati;
- `backgroundButton`, colore di sfondo dei pulsanti abilitati;
- `overlayButton`, colore di sfondo dei pulsanti abilitati quando vengono premuti;
- `confirmButton`, colore di sfondo dei pulsanti di conferma;
- `deleteButton`, colore di sfondo dei pulsanti di annullamento o eliminazione.

#### ButtonsPadding

`ButtonsPadding` è un classe privata, infatti verrà utilizzata solamente all'interno del file per la configurazione dello stile grafico dei pulsanti, in cui ne viene ridefinito il `padding` [49].

Ne sono stati implementati due: `normalButtonPadding` e `smallButtonPadding`, il primo è stato applicato alla maggior parte dei pulsanti presenti nell'applicazione, mentre il secondo ai pulsanti contenuti in componenti di dimensioni ridotte (es. `AlertDialog`).

#### ButtonsBorder

`ButtonsBorder` è una classe privata implementata per definire la proprietà `border` [6], in particolare è stato semplicemente impostato il raggio degli angoli a 20.0.

#### ElevatedButton

`ElevatedButton` è una classe pubblica che contiene tutte le impostazioni grafiche per i pulsanti di tipo `ElevatedButton` [18], in particolare i colori che deve assumere in base agli stati di attivazione e disattivazione, il `padding`, precisamente `normalButtonPadding`, e `border radius` [6].

Questo viene fatto, oltre per lo stile del pulsante base, ma anche per quelli di conferma e di annullamento/eliminazione.

### AlertDialogButton

`AlertDialogButton` è una classe che definisce lo stile grafico dei pulsanti impiegati nelle finestre di dialogo, in particolare ne vengono configurati solo due, uno per la conferma e uno per l'annullamento.

Entrambi condividono lo stesso *padding*, precisamente `smallButtonPadding`, e *border radius* [6], mentre differiscono per il colore di sfondo, `confirmButton` e `deleteButton` rispettivamente e per la tipologia.

### 5.7.3 TextStyles

Nel file `text_styles.dart` sono presenti differenti classi, il cui scopo è di ridefinire lo stile grafico dei testi utilizzati nell'applicazione.

La classe principale `TextStyles` contiene vari stili di testo, in particolare per ciascuno di essi è stato definito: `fontSize` [30] e `fontWeight` [31].

Gli stili di testo [71] in questione sono: `displayLarge`, `displayMedium`, `displaySmall`, `titleLarge`, `titleMedium`, `titleSmall`, `bodyLarge`, `bodyMedium` e `bodySmall`.

È presente un ulteriore classe, `AlertDialogTextStyles`, in cui viene ridefinito lo stile grafico dei testi utilizzati nelle finestre di dialogo, in particolare una configurazione per il titolo, `alertDialogTitleStyle`, e una per il contenuto, `alertDialogContentStyle`, richiamando per ciascuno, uno degli stili sopra elencati.

### 5.7.4 AppTheme

`AppTheme` è una classe che si occupa di ridefinire il tema dell'applicazione, richiamando per ciascuna delle proprietà di `ThemeData` [73] le classi precedentemente descritte.

I colori precedentemente configurati sono stati richiamati nella proprietà `colorScheme`, mentre lo stile dei testi in `textTheme`.

Nelle proprietà `elevatedButtonTheme` e `outlinedButtonTheme` sono stati richiamati rispettivamente le configurazioni grafiche illustrate in precedenza.

Infine è stato impostato *ad hoc* lo stile grafico del `floatingActionButton` [22].

## 5.8 Utils

Nella cartella `utils` sono raccolti file, in cui sono definite delle classi che espongono metodi ausiliari per la gestione di determinate funzionalità.

Di seguito, per ciascun file, verranno illustrate nel dettaglio le classi presenti.

### 5.8.1 Biometric auth

La classe `BiometricAuth` implementa la libreria *local auth* [43] per la gestione dell'autenticazione attraverso il riconoscimento biometrico.

Vi sono presenti due metodi: `isBiometricSupported` che ritorna `true`, se il dispositivo supporta il riconoscimento biometrico e `authenticate` che si occupa di effettuare l'autenticazione attraverso il riconoscimento biometrico, se l'operazione va a buon fine, viene restituito `true`, altrimenti `false` e in quest'ultimo caso è comunque possibile effettuare l'autenticazione attraverso l'inserimento manuale delle credenziali.

### 5.8.2 Debouncer

`Debouncer` è una classe che implementa il design pattern [g] *debouncer*: tecnica per prevenire l'esecuzione di una funzione molteplici volte.

In questo progetto è servito per la gestione della ricerca dell'identificativo dei clienti, evitando di effettuare una richiesta HTTP [g] al back-end [g] ad ogni carattere digitato dall'utente, inserendo dunque un intervallo di tempo per ritardarne l'esecuzione.

### 5.8.3 Exception handler

La classe `UnauthorizedException` implementa l'interfaccia `Exception` [19], per gestire le eccezioni che si verificano quando l'utente non è autorizzato ad effettuare una determinata richiesta al back-end [g].

### 5.8.4 Filtering

La classe `FilteringOptions` incapsula le informazioni necessarie per effettuare un filtraggio nella lista di *Meeting Note*.

Le variabili di stato sono:

- `object`, `MeetingNoteObject` che rappresenta il cliente [g];
- `dateRange`, `DateTimeRange` che rappresenta l'intervallo di date;
- `isChronologicalOrder`, `bool` che indica se l'ordinamento deve avvenire in maniera cronologica o meno.

Inoltre sono presenti, dei metodi per impostare il valore di ciascuna variabile di stato e un metodo `isNull` che restituisce `false` se tutte le variabili di stato sono `null`.

È stato creato poi un *Provider*, di tipo `StateNotifierProvider` [63], per gestire lo stato di `FilteringOptions` nella schermata principale dell'applicazione (vedi Sezione 5.6.2).

### 5.8.5 Network handler

Nel file `network_handler.dart` è contenuto una enumerazione di tipo `NetworkStatus` che rappresenta lo stato della connessione ad internet, che può essere assunto dall'applicazione, in particolare: `notDetermined`, `connected` e `disconnected`.

È presente la classe `NetworkDetectorNotifier` in cui è stato implementato un costruttore che si occupa di inizializzare la variabile di stato `networkStatus` con il valore `notDetermined` e di aggiornarla quando avvengono dei cambiamenti.

È stato creato poi `networkAwareProvider` un *provider*, di tipo `StateNotifierProvider` [63], per leggere lo stato della connessione ad internet a livello globale.

### 5.8.6 User preferences

Nel file `user_preferences.dart` vengono implemetate le librerie `SharedPreferences` [59] e `SecureStorage` [28] per la memorizzazione di alcuni dati in locale, a supporto delle funzionalità di autenticazione e riconoscimento biometrico.

Si specifica che sono state definite due classi, per entrambe le due librerie, utilizzando il design pattern [g] singleton [g], in quanto è necessario che vi sia una sola istanza per ciascuna classe.

La classe `AuthPreferences` ha lo scopo di gestire il *token* di autenticazione e lo stato di abilitazione del riconoscimento biometrico, esponendo i seguenti metodi:

- `saveToken`, salva il *token* di autenticazione;
- `getToken`, restituisce il *token* di autenticazione;
- `deleteToken`, elimina il *token* di autenticazione.
- `saveBiometricState`, salva una variabile *booleana* che indica se l'utente ha abilitato il riconoscimento biometrico;
- `getBiometricState`, restituisce il valore della variabile *booleana* che indica se l'utente ha abilitato il riconoscimento biometrico.

Mentre la classe `SecureStorage` ha lo scopo di gestire le credenziali di accesso, esponendo i seguenti metodi:

- `setUsername`, cifra e salva l'username;
- `getUsername`, decifra e restituisce l'username;
- `removeUsername`, elimina l'username;
- `setPassword`, cifra e salva la password;
- `getPassword`, decifra e restituisce la password;
- `removePassword`, elimina la password;
- `hasCredentials`, restituisce `true` se sono salvate e cifrate le credenziali di accesso.
- `removeAll`, elimina tutte le credenziali di accesso, invocando i metodi `removeUsername` e `removePassword`;
- `saveAll`, salva e cifra le credenziali di accesso, invocando i metodi `setUsername` e `setPassword`.

### 5.8.7 Show alert

Nel file `show_alert.dart` sono presenti dei metodi pubblici che si occupano di visualizzare all'utente delle finestre di dialogo o *snackbar* [60], per mostrare messaggi di errore o di conferma (Figura 5.26).

È stato implementato un metodo, che funge da base e quindi personalizzabile, per la visualizzazione delle finestre di dialogo discusse nella Sezione 5.2.1.

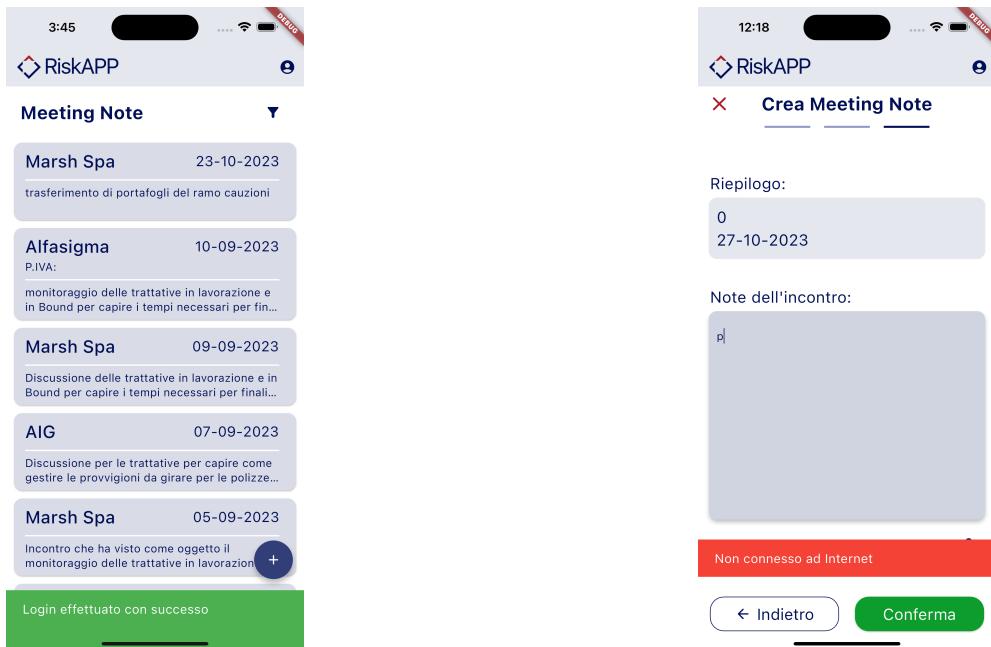
In aggiunta è presente un metodo per visualizzare una *snackbar*, la quale può essere personalizzata in base al messaggio da mostrare e al colore di sfondo.

### 5.8.8 Speech to text

Nel file `speech_to_text.dart` è presente la classe `AudioRecognizer` che implementa la libreria `SpeechToText` [61] per la gestione della dettatura vocale.

In essa sono stati definiti i seguenti metodi:

- `initSpeech`, determina se il dispositivo supporta la dettatura vocale;
- `startListening`, avvia la dettatura vocale, inoltre vengono impostati alcuni parametri, come la lingua e la durata massima della registrazione;
- `stopListening`, interrompe la dettatura vocale;
- `isListening`, restituisce `true` se la dettatura vocale è in corso;



**Figura 5.26:** Esempi di *snackbar*

## 5.9 main.dart

Il file `main.dart` è il punto di ingresso dell'applicazione, in particolare è presente la funzione `main` che si occupa di inizializzare l'applicazione, fissando l'orientamento dello schermo a `portraitUp` e dichiarando il `ProviderScope`.

Richiama poi `MyApp`, un *ConsumerWidget*, e come prima cosa controlla se è salvato in memoria il *token* di autenticazione attraverso `authProvider` (vedi Sezione 5.5.1) e decidere se rimandare alla schermata di autenticazione o alla schermata principale dell'applicazione.

Il reindirizzamento avviene attraverso `InitApp`, un *StatelessWidget*, invocando `ScreenUtilInit` (vedi Sezione 5.10), fissando inoltre il parametro `supportedLocales` a `const Locale('it', 'IT')`, in modo tale che l'applicazione imposti automaticamente alcuni *widget* come il calendario e la data nel formato corretto e infine viene impostato il tema dell'applicazione (discusso precedentemente).

## 5.10 Responsività

Per la gestione della responsività dell'applicazione, è stata utilizzata la libreria `flutter_screenutil` [55] che si occupa di adattare le dimensioni degli elementi grafici in base a quelle dello schermo del dispositivo.

In particolare è stato invocato `ScreenUtilInit` per inizializzare la libreria, fissando il parametro `designSize` con le dimensioni definite nel mockup [g], in modo tale che la libreria possa mantenere le proporzioni degli elementi grafici.

## 5.11 Documentazione

### README

Insieme al codice sorgente dell'applicazione, è stata consegnata anche una breve documentazione, scritta in lingua inglese, secondo la richiesta dell'azienda, utilizzando il linguaggio Markdown [g].

È stato dunque redatto un file `README.md`, il cui contenuto è riportato di seguito:

- **Abstract:** breve descrizione dell'applicazione e del contesto applicativo;
- **Getting Started:** istruzioni per poter compilare ed eseguire l'applicazione;
- **Libraries Used:** elenco delle librerie utilizzate, necessarie per la compilazione;
- **Project Structure:** descrizione della struttura e architettura del progetto;
- **Features:** elenco delle funzionalità implementate.

### Documentazione del codice

Per quanto riguarda il codice sorgente, è stato scritto un commento per ogni classe, metodo e variabile, in modo da rendere più chiara la lettura del codice.

Lo standard seguito è quello proposto nella documentazione ufficiale di *Dart* [11].

In particolare, per i commenti alle classi, è stato utilizzato il formato, illustrato nel Listing 5.1, spiegando anche il significato e scopo delle variabili di stato.

```
//> Class encapsulating the authentication's params,
//> given in input to the authentication request.
//>
//> [<code>_username</code>], the user's username
//>
//> [<code>_password</code>], the user's password
class AuthArgs {
    final String _username;
    final String _password;
    // ...
}
```

**Listing 5.1:** Commento classe

Per i commenti ai metodi, è stato utilizzato il medesimo formato utilizzato per i commenti alle classi, illustrato nel Listing 5.2, spiegando lo scopo del metodo, dei parametri in input e il valore di ritorno.

```
/// Authenticate the user with the given authentication [  
    args]  
/// retrieving the token and saving it in the local storage  
    if the authentication request is successful.  
///  
/// Return [AuthResponse] object with:  
///  
/// - [AuthValues] object, containing the token if the  
    request is successful  
///  
/// - [StatusResponse] object, containing the response's  
    status  
Future<AuthResponse> authenticate({required AuthArgs args})  
    async {  
        log("credentials: ${args.toString()}");  
  
        // ...  
  
        log("auth response: ${result.toString()}");  
  
        // ...  
    }
```

**Listing 5.2:** Commento metodo

# Capitolo 6

## Conclusioni

*In questo ultimo capitolo verranno illustrati gli obiettivi raggiunti, il consuntivo delle attività e infine verrà fornita una valutazione personale sul tirocinio svolto.*

### 6.1 Obiettivi raggiunti

La Tabella 6.1 illustra tutti gli obiettivi raggiunti al termine del tirocinio, con l'eccezione di un obiettivo desiderabile e di uno facoltativo.

Il mancato soddisfacimento degli obiettivi *D02* e *F01*, ovvero il *deploy* e la stesura dei test per la verifica e validazione dell'applicazione, è stato causato principalmente dalla mancanza di tempo.

Come evidenziato dal reale svolgimento delle attività (Sezione 6.2), la fase di sviluppo è stata più lunga del previsto.

Malgrado non siano stati progettati e scritti i test per la verifica e validazione dell'applicazione, si è comunque cercato di mantenere un codice pulito e ben strutturato, in modo da facilitare eventuali modifiche e manutenzioni future, anche se come garanzia di qualità non può considerarsi sufficiente. Si specifica, inoltre, che la verifica delle funzionalità dell'applicazione è stata effettuata dal tutor aziendale o da altri colleghi, lungo tutto il periodo di sviluppo del progetto, fornendo un *feedback* immediato, permettendo così di apportare eventuali modifiche correttive e miglioramenti.

Per quanto riguarda il *deploy*<sup>[gl]</sup>, il principale motivo per cui non è stato effettuato, è da imputarsi al fatto che l'utenza a cui è rivolta l'applicazione è molto ristretta, risultava quindi superfluo e non necessario pubblicarla sulle piattaforme *Google Play Store* e *Apple App Store*. Tale decisione è stata presa dal tutor aziendale che ha ritenuto più opportuno pensare ad un modo alternativo per distribuire l'applicazione direttamente agli utenti finali.

### 6.2 Consuntivo delle attività

Durante la prima settimana di stage, la pianificazione iniziale non è stata rispettata sin da subito, dunque la reale ripartizione delle attività ha subito delle differenze, questo però non ha compromesso negativamente in alcun modo il percorso di realizzazione del prodotto e il raggiungimento degli obiettivi fissati.

Una prima motivazione di tali variazioni è dettata da un consiglio del tutor aziendale,

Obiettivo	Descrizione	Priorità	Stato
O01	Implementazione di meccanismi di autenticazione	Obbligatorio	Soddisfatto
O02	Richiamare i dati dal backend e visualizzarli all'interno dell'applicazione	Obbligatorio	Soddisfatto
O03	Implementazione di funzionalità per la creazione delle <i>Meeting Note</i>	Obbligatorio	Soddisfatto
O04	Implementazione di funzionalità per la compilazione delle <i>Meeting Note</i>	Obbligatorio	Soddisfatto
O05	Sviluppo interfaccia grafica dell'applicazione	Obbligatorio	Soddisfatto
D01	Integrazione del servizio OpenAI per automatizzare la creazione di <i>Meeting Note</i>	Desiderabile	Soddisfatto
D02	Deploy dell'applicazione	Desiderabile	Non soddisfatto
F01	Esecuzione dei test per la verifica e validazione dell'applicazione	Facoltativo	Non soddisfatto

Tabella 6.1: Tabella con gli obiettivi raggiunti

ovvero quello di sviluppare un mockup<sup>[g]</sup> dell'applicazione, in modo da fornire una notevole agevolazione nell'analisi dei requisiti e di conseguenza avere un'idea più chiara di come essa si sarebbe dovuta progettare e sviluppare.

Un'altra motivazione, è stata la mia esperienza pregressa, da autodidatta, con il linguaggio *Dart* [14] e il framework *Flutter* [23], che mi ha permesso di saltare parzialmente parte della fase formativa.

Tali variazioni, corrispondenti dunque all'effettivo svolgimento delle attività, sono visualizzabili nel diagramma di Gantt in Figura 6.1, il cui consuntivo delle ore corrispondente è riportato nella Tabella 6.2.

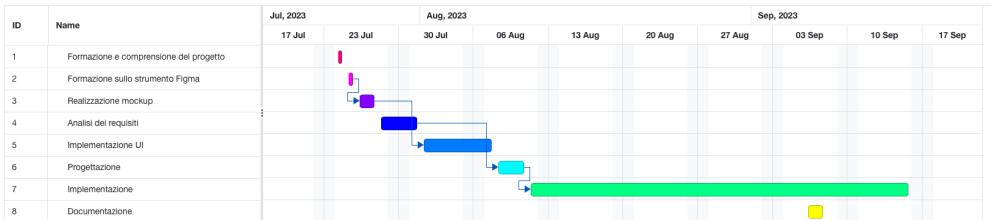


Figura 6.1: Diagramma di Gantt della ripartizione definitiva delle attività durante il periodo di stage

È doveroso precisare che, nonostante la fase di progettazione sia avvenuta dopo la codifica della UI<sup>[g]</sup> dell'applicazione e non viceversa, come previsto nella pianificazione iniziale, ma soprattutto dai principi dell'ingegneria del software<sup>[g]</sup>, non si è andati a compromettere la fase di sviluppo, in quanto la codifica della UI<sup>[g]</sup> si è svolta con la consapevolezza di implementare un'architettura che si basasse sul pattern architettonico<sup>[g]</sup> MVC<sup>[g]</sup>, per mantenere una separazione tra la logica di presentazione e la logica di business.

Ore effettive	Ore pianificate	Descrizione dell'attività
<b>70</b>	<b>80</b>	<b>Formazione sulle tecnologie</b>
<b>40</b>	<b>40</b>	<b>Definizione architettura di riferimento e relativa documentazione</b>
14 20 6	12 22 6	Analisi del problema e del dominio applicativo Progettazione della piattaforma e relativi test Stesura documentazione relativa ad analisi e progettazione
<b>192</b>	<b>160</b>	<b>Sviluppo</b>
40	26	Implementazione del meccanismo di autenticazione e integrazione con il sistema preesistente per il prelevamento dei dati da visualizzare nell'applicazione
40	34	Implementazione delle funzionalità di creazione e compilazione (testuale e vocale) di Meeting Note
40	28	Integrazione del servizio OpenAI, per la creazione automatica di Meeting Note
24	40	Attività di testing
48	32	Sviluppo UI
<b>10</b>	<b>40</b>	<b>Verifica e Validazione finale</b>
0	24	Esecuzione dei test per la verifica e collaudo dell'applicazione
10	8	Stesura documentazione finale
0	8	Deploy dell'applicazione
<b>312</b>		<b>Totale ore</b>

Tabella 6.2: Consuntivo della durata delle attività

Infine, per quanto riguarda l'attività di *Integrazione del servizio OpenAI per automatizzare la creazione di Meeting Note*, è stato addestrato e utilizzato un modello per effettuare elaborazione del testo: a partire da un testo in input, digitato dall'utente, tale modello deve essere in grado di estrarre e restituire le informazioni rilevanti per la creazione di una Meeting Note<sup>[g]</sup>, ovvero il nome del cliente, la data e il contenuto dell'incontro.

L'implementazione di tale modello è stata effettuata dal tutor aziendale, il quale ha successivamente integrato questa funzionalità nel back-end<sup>[g]</sup> della piattaforma *RiskAPP*. Per l'integrazione di tale servizio nell'applicazione, si è reso sufficiente effettuare una richiesta HTTP<sup>[g]</sup> al back-end<sup>[g]</sup>, come descritto nelle Sezioni 5.4.1 e 5.5.3.

Dunque, le ore effettive per questa attività sono da intendersi come il tempo impiegato per l'integrazione del servizio, presente nel back-end<sup>[g]</sup>, nell'applicazione e non per l'implementazione del modello.

### 6.3 Valutazione personale

Il tirocinio è stato un'esperienza molto formativa, in quanto mi ha permesso di mettere in pratica le conoscenze e metodologie acquisite durante il percorso di studi in un contesto lavorativo.

Come già menzionato in precedenza, nonostante la mia familiarità con alcune delle tecnologie adottate, ho avuto modo di approfondirle e di imparare nuovi concetti, soprattutto per quanto riguarda il *framework Flutter*.

Inoltre, per la prima volta ho avuto modo di lavorare ad un progetto in maniera più professionale, seguendo anche alcuni principi affrontati durante il corso di *Ingegneria del Software*, come ad esempio la pianificazione della fase di sviluppo, fissando delle milestone<sup>[gl]</sup>, mantenendo traccia dei progressi per il raggiungimento degli obiettivi del progetto, e la documentazione del codice, in modo da rendere più facile la comprensione e la manutenzione del codice.

L'azienda mi ha lasciato molta autonomia durante lo sviluppo del progetto e mi ha fornito un *feedback* costante, permettendomi di migliorare e di imparare nuove cose. In conclusione, ritengo che il tirocinio sia stato un'esperienza molto positiva, che mi ha permesso di crescere sia dal punto di vista professionale che personale.

# Acronimi

**API** Application Program Interface<sup>[g]</sup>. 4, 5, 92, 94

**HTTP** HyperText Transfer Protocol<sup>[g]</sup>. 93

**IA** Intelligenza Artificiale<sup>[g]</sup>. 93

**JSON** JavaScript Object Notation<sup>[g]</sup>. 94

**MVC** Model View Controller<sup>[g]</sup>. 94

**UI** User Interface<sup>[g]</sup>. 95

**UML** Unified Modeling Language<sup>[g]</sup>. 92, 95

**URI** Uniform Resource Identifier<sup>[g]</sup>. 93, 95

# Glossario

Questo glossario contiene le definizioni dei termini tecnici, degli acronimi e delle abbreviazioni utilizzati nel documento.

Per la definizione della maggioranza di questi termini, è stata utilizzata un'unica fonte di riferimento: *Wikipedia* [75].

Per i restanti termini, verranno indicate, per ciascuno, le fonti di riferimento.

**API** in informatica, con il termine *Application Programming Interface API* (ing. interfacce di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 4, 5, 7, 36, 38, 42, 57, 58, 61, 62, 91

**Attore** in informatica, con il termine *attore* si indica un ruolo che l'utente o un sistema esterno ha nell'interazione con il sistema in esame. 7, 92

**Back-end** in informatica, si indica la parte di un sistema software non accessibile direttamente dall'utente, tipicamente responsabile della gestione dei dati e della loro elaborazione. 7, 36, 42, 57–60, 62, 63, 65, 66, 69, 73, 75, 77, 78, 82, 89

**Baseline** termine che viene utilizzato nella disciplina di *ingegneria del software* per indicare un punto di avanzamento consolidato da cui ripartire per proseguire lo sviluppo del progetto. 94

**Caso d'uso** insieme di scenari (sequenza di azioni) che hanno in comune un obiettivo finale per un attore [8].

Sono rappresentabili attraverso dei 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. 7, 8, 26, 31

**Cliente** nel contesto applicativo della tesi, costituisce l'oggetto di una *Meeting Note* e può essere di tre tipologie: **broker**, **contraenti** o **compagnia assicurativa**.

Il primo tra questi è un intermediario tra il cliente finale e la compagnia assicurativa, mentre i secondi sono aziende che sottoscrivono un contratto della polizza assicurativa. 2, 3, 9, 12, 18, 23, 24, 26–30, 59–61, 64, 66, 72–74, 76, 82, 94

**Cross-platform** in informatica, con il termine *cross-platform* (ing. multi-piattaforma) si indica un software che può essere eseguito su più di un sistema hardware o software. Il termine si riferisce in particolare a sistemi operativi, ma anche a software applicativi, linguaggi di programmazione e strumenti di sviluppo. 2, 3

**Deploy** termine che indica la fase di distribuzione di un software, ovvero l'insieme di tutte le attività necessarie per rendere disponibile il prodotto finito all'utilizzatore finale. 4, 6, 87

**Design pattern** in informatica, con il termine *design pattern* (ing. schema progettuale) si indica una soluzione che può essere applicata in più contesti, per risolvere un problema ricorrente. 82, 95

**Endpoint** in informatica, con il termine *endpoint* si indica un'interfaccia esposta da un servizio web, attraverso la quale è possibile interagire con esso. 57

**Framework** in informatica, con il termine *framework* si indica un'architettura logica di supporto su cui un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore. 37, 41

**Funnel commerciale** modello di marketing che consiste in un processo suddiviso in più fasi, attraverso le quali si cerca di convertire un utente generico in un cliente effettivo. 2

**Gesture** si indica un insieme di azioni che l'utente può compiere attraverso l'interazione con il dispositivo mobile, come ad esempio, toccare, trascinare, pizzicare, ecc.. 38

**Git** sistema di controllo versione distribuito, utilizzato per il tracciamento delle modifiche al codice sorgente durante lo sviluppo del software. 93

**GitHub** piattaforma, basata sul sistema di controllo versione Git<sup>[g]</sup>, che permette di controllare e versionare il codice sorgente.  
Offre molte altre funzionalità, tra cui la condivisibilità del codice e segnalazione di problematiche. 35

**HTTP** con il termine *HTTP, HyperText Transfer Protocol* (ing. protocollo di trasferimento ipertesto) si indica un protocollo a livello applicativo usato come principale sistema per la trasmissione d'informazioni sul web, ovvero in un'architettura tipica client-server.

Le richieste sono composte da un URI, versione del protocollo e il metodo, che indica l'azione da compiere. Le risposte contengono un codice di stato, informazioni sul server e sulla risorsa richiesta e il contenuto della risorsa stessa. 4, 7, 36, 39, 58, 62, 63, 66, 82, 89, 91

**Intelligenza Artificiale** insieme di teorie e tecniche che permettono a un sistema di presentare un comportamento intelligente, ovvero di emulare l'intelligenza umana. 1, 22, 34, 60, 91

**Ingegneria del software** disciplina che si occupa di tutti gli aspetti della produzione del software, dalla fase di progettazione a quella di manutenzione, passando per la codifica, la verifica e la validazione, fornendo principi e metodologie da applicare con lo scopo di garantire efficacia ed efficienza nel processo di sviluppo. 88

**Issue tracking** si intende un sistema che permette di tenere traccia delle problematiche, riscontri, idee o attività durante lo sviluppo del software, in modo da poterle gestire in maniera efficiente. 41

**JSON** con il termine *JSON* si indica un formato adatto all'interscambio di dati fra applicazioni client-server. 58–60, 65, 91

**Markdown** linguaggio utilizzato per la formattazione di testo semplice, in particolare per la scrittura di documentazione tecnica. 85

**Meeting Note** nel contesto applicativo della tesi, si tratta di una nota compilabile dall'utente per monitorare gli incontri con i clienti. Le informazioni che contiene sono: il nome identificativo del **Cliente**<sup>[g]</sup>, la **Data** dell'incontro, il **Contenuto** e l'**Autore**. 2–5, 9, 11–13, 17–22, 24–27, 29, 30, 89

**Milestone** termine che viene utilizzato nella disciplina di *ingegneria del software* per indicare un punto di riferimento nel tempo che indica un obiettivo da raggiungere, ovvero un evento che rappresenta un traguardo parziale nello svolgimento del progetto, deve essere consolidato dal raggiungimento di una o più baseline<sup>[g]</sup>. 41, 90

**Mockup** indica una realizzazione a scopo illustrativo di un prodotto, con il fine di fornire in anteprima una rappresentazione del prodotto stesso, mancante dell'implementazione effettiva delle sue funzionalità, ma che ne mostri ugualmente le caratteristiche salienti. 5, 37, 41, 50, 79, 84, 88

**MVC** indica il pattern architettonale<sup>[g]</sup> Model-View-Controll, che separa la logica di presentazione dei dati dalla logica di business.

Il *modello* rappresenta la struttura dei dati dell'applicazione e le regole che governano la loro manipolazione.

La *vista* è la rappresentazione visuale del modello.

Il *controllore* è il componente che gestisce gli eventi, che possono essere generati sia dalla vista che dal modello, e aggiorna il modello e la vista. 39, 88, 91

**Open source** in informatica, con il termine *open source* si indica un software di cui il codice sorgente è pubblico, in modo che esso possa essere studiato, modificato e utilizzato da chiunque. 38, 41

**OpenAI API** [46] insieme di servizi e risorse messi a disposizione da *OpenAI*, che permettono agli sviluppatori di integrare modelli di intelligenza artificiale nei loro prodotti software. 4, 5

**Overriding** in informatica, con il termine *overriding* si indica la possibilità di ridefinire un metodo ereditato da una classe base, in modo da adattarlo alle esigenze della classe derivata. 58

**Padding** [49] in *Flutter*, indica un margine interno che viene applicato ad un *widget*, definisce la distanza tra il contenuto e il suo bordo. 44

**Pattern architetturale** in informatica, con il termine *pattern architetturale* si intende una soluzione, riutilizzabile, a un problema di progettazione architettonica, in un contesto definito.

Delinea una struttura di base per un sistema software, specificando le sue parti, le loro responsabilità e le relazioni tra di esse. 88, 94

**Responsività** in informatica, con il termine *responsività* si indica la capacità di un sistema di adattarsi a diverse situazioni, in particolare, nel contesto applicativo, si intende la capacità di un'applicazione di adattarsi a diverse dimensioni dello schermo dei dispositivi mobile. 35

**Riconoscimento biometrico** si intende una particolare forma di autenticazione attraverso caratteristiche fisiologiche e/o comportamentali dell'utente, come ad esempio, tra quelle fisiologiche più comuni, l'impronta digitale e il riconoscimento facciale. 3, 8, 9, 15, 16

**Singleton** si indica un design pattern<sup>[8]</sup> creazionale con lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza. 82

**UI** in informatica, con il termine *UI, User Interface* (ing. interfaccia utente) si indica la parte del software che permette l'interazione con l'utente, attraverso l'uso di dispositivi hardware e software.

L'interfaccia utente può essere grafica (Graphical User Interface - GUI) o a linea di comando (Command Line Interface - CLI). 4, 5, 37, 38, 88, 91

**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. 42, 91

**URI** in informatica, con il termine *URI, Uniform Resource Identifier* (ing. identificatore uniforme di risorsa) si indica una sequenza di caratteri che identifica univocamente una risorsa generica che può essere un indirizzo Web, un documento, un'immagine, un file, un servizio, ecc.. 57, 63–65, 91

**Wizard** in informatica, con il termine *wizard* si indica un programma, solitamente contenuto in un'applicazione più complessa, che permette all'utente, attraverso una procedura passo-passo, di essere guidato per l'esecuzione di un compito, con lo scopo di semplificare l'interazione con il software. 3, 53, 54, 74, 75, 77, 78

# Riferimenti bibliografici

## Siti web consultati

- [1] *AlertDialog class*. URL: <https://api.flutter.dev/flutter/material/AlertDialog-class.html> (cit. a p. 44).
- [2] *Android Studio*. URL: <https://developer.android.com/studio> (cit. a p. 42).
- [3] *AppBar class*. URL: <https://api.flutter.dev/flutter/material/AppBar-class.html> (cit. a p. 45).
- [4] *Autocomplete class*. URL: <https://api.flutter.dev/flutter/material/Autocomplete-class.html> (cit. a p. 55).
- [5] *AvatarGlow*. URL: [https://pub.dev/packages/avatar\\_glow](https://pub.dev/packages/avatar_glow) (cit. a p. 52).
- [6] *Border class*. URL: <https://api.flutter.dev/flutter/painting/Border-class.html> (cit. alle pp. 80, 81).
- [7] *Card class*. URL: <https://api.flutter.dev/flutter/material/Card-class.html> (cit. a p. 80).
- [8] *Chips*. URL: <https://m3.material.io/components/chips/overview> (cit. a p. 79).
- [9] *CircleAvatar class*. URL: <https://api.flutter.dev/flutter/material/CircleAvatar-class.html> (cit. a p. 68).
- [10] *CircularProgressIndicator class*. URL: <https://api.flutter.dev/flutter/material/CircularProgressIndicator-class.html> (cit. alle pp. 44, 67, 71).
- [11] *Comments*. URL: <https://dart.dev/language/comments#documentation-comments> (cit. a p. 85).
- [12] *CupertinoPicker class*. URL: <https://api.flutter.dev/flutter/cupertino/CupertinoPicker-class.html> (cit. a p. 50).
- [13] *Dart extension*. URL: <https://marketplace.visualstudio.com/items?itemName=Dart-Code.dart-code> (cit. a p. 41).
- [14] *Dart programming language*. URL: <https://dart.dev/> (cit. alle pp. 5, 36, 88).
- [15] *DateTime class*. URL: <https://api.flutter.dev/flutter/dart-core/DateTime-class.html> (cit. a p. 59).
- [16] *Divider class*. URL: <https://api.flutter.dev/flutter/material/Divider-class.html> (cit. alle pp. 50, 54).

- [17] *DropdownButton class*. URL: <https://api.flutter.dev/flutter/material/DropdownMenu-class.html> (cit. a p. 50).
- [18] *ElevatedButton class*. URL: <https://api.flutter.dev/flutter/material/ElevatedButton-class.html> (cit. alle pp. 48, 71, 75, 77, 78, 80).
- [19] *Exception class*. URL: <https://api.flutter.dev/flutter/dart-core/Exception-class.html> (cit. a p. 82).
- [20] *Factory constructors*. URL: <https://dart.dev/language/constructors#factory-constructors> (cit. a p. 58).
- [21] *Figma*. URL: <https://www.figma.com/> (cit. alle pp. 5, 41).
- [22] *FloatingActionButton class*. URL: <https://api.flutter.dev/flutter/material/FloatingActionButton-class.html> (cit. alle pp. 69, 81).
- [23] *Flutter*. URL: <https://docs.flutter.dev> (cit. alle pp. 5, 36, 41, 79, 88).
- [24] *Flutter App Architecture with Riverpod: An Introduction*. URL: <https://codewithandrea.com/articles/flutter-app-architecture-riverpod-introduction/> (cit. a p. 39).
- [25] *Flutter architectural overview*. URL: <https://docs.flutter.dev/resources/architectural-overview> (cit. alle pp. 38, 39).
- [26] *Flutter extension*. URL: <https://marketplace.visualstudio.com/items?itemName=Dart-Code.flutter> (cit. a p. 41).
- [27] *Flutter Project Structure: Feature-first or Layer-first?* URL: <https://codewithandrea.com/articles/flutter-project-structure/#:~:text=Layer%2Dfirst%3A%20Drawbacks&text=And%20if%20we%20decide%20that,when%20building%20medium%2Flarge%20apps.> (cit. a p. 40).
- [28] *flutter secure storage*. URL: [https://pub.dev/packages/flutter\\_secure\\_storage](https://pub.dev/packages/flutter_secure_storage) (cit. a p. 82).
- [29] *Flutter state management*. URL: <https://docs.flutter.dev/data-and-backend/state-mgmt> (cit. a p. 38).
- [30] *fontSize property*. URL: <https://api.flutter.dev/flutter/painting/TextStyle/fontSize.html> (cit. a p. 81).
- [31] *fontWeight property*. URL: <https://api.flutter.dev/flutter/painting/TextStyle/fontWeight.html> (cit. a p. 81).
- [32] *FutureBuilder class*. URL: <https://api.flutter.dev/flutter/widgets/FutureBuilder-class.html> (cit. a p. 67).
- [33] *FutureProvider*. URL: [https://riverpod.dev/docs/providers/future\\_provider](https://riverpod.dev/docs/providers/future_provider) (cit. alle pp. 39, 65).
- [34] *Git*. URL: <https://git-scm.com/> (cit. a p. 41).
- [35] *GitHub*. URL: <https://docs.github.com/get-started> (cit. a p. 41).
- [36] *hashCode property*. URL: <https://api.flutter.dev/flutter/dart-core/Object/hashCode.html> (cit. a p. 58).

- [37] *Icon class.* URL: <https://api.flutter.dev/flutter/widgets/Icon-class.html> (cit. a p. 56).
- [38] *IconButton class.* URL: <https://api.flutter.dev/flutter/material/IconButton-class.html> (cit. alle pp. 52, 53).
- [39] *IconData class.* URL: <https://api.flutter.dev/flutter/widgets/IconData-class.html> (cit. a p. 44).
- [40] *Infinite scroll pagination.* URL: [https://pub.dev/packages/infinite\\_scroll\\_pagination](https://pub.dev/packages/infinite_scroll_pagination) (cit. alle pp. 68, 69, 75).
- [41] *Inherited Widget.* URL: <https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html> (cit. a p. 39).
- [42] *ListTile class.* URL: <https://api.flutter.dev/flutter/material/ListTile-class.html> (cit. alle pp. 50, 76).
- [43] *local-auth.* URL: <https://pub.dev/packages/local-auth> (cit. a p. 81).
- [44] *Mixins.* URL: <https://dart.dev/language/mixins> (cit. a p. 63).
- [45] *onPressed property.* URL: <https://api.flutter.dev/flutter/material/MaterialButton/onPressed.html> (cit. alle pp. 47, 52).
- [46] *OpenAI.* URL: <https://openai.com> (cit. a p. 94).
- [47] *operator ==.* URL: [https://api.flutter.dev/flutter/dart-core/Object/operator\\_equals.html](https://api.flutter.dev/flutter/dart-core/Object/operator_equals.html) (cit. a p. 58).
- [48] *OutlineButton class.* URL: <https://api.flutter.dev/flutter/material/OutlinedButton-class.html> (cit. alle pp. 73, 77, 78, 80).
- [49] *Padding class.* URL: <https://api.flutter.dev/flutter/widgets/Padding-class.html> (cit. alle pp. 80, 94).
- [50] *Provider.* URL: <https://pub.dev/packages/provider> (cit. a p. 38).
- [51] *Reading a Provider.* URL: <https://riverpod.dev/docs/concepts/reading> (cit. alle pp. 39, 47).
- [52] *RiskAPP.* URL: <https://www.riskapp.it/riskapp-platform> (cit. alle pp. 1, 2).
- [53] *Riverpod.* URL: <https://riverpod.dev/> (cit. alle pp. 38, 39).
- [54] *Scaffold class.* URL: <https://api.flutter.dev/flutter/material/Scaffold-class.html> (cit. a p. 53).
- [55] *screenutil.* URL: [https://pub.dev/packages/flutter\\_screenutil](https://pub.dev/packages/flutter_screenutil) (cit. a p. 84).
- [56] *Scroll Date Picker.* URL: [https://pub.dev/packages/scroll\\_date\\_picker](https://pub.dev/packages/scroll_date_picker) (cit. a p. 54).
- [57] *Sealed class.* URL: <https://dart.dev/language/class-modifiers#sealed> (cit. a p. 61).
- [58] *SearchBar class.* URL: <https://api.flutter.dev/flutter/material/SearchBar-class.html> (cit. a p. 54).

- [59] *shared preferences*. URL: [https://pub.dev/packages/shared\\_preferences](https://pub.dev/packages/shared_preferences) (cit. a p. 82).
- [61] *speech-to-text*. URL: <https://pub.dev/packages/speech-to-text> (cit. alle pp. 71, 83).
- [62] *StarUML*. URL: <https://staruml.io> (cit. a p. 42).
- [63] *StateNotifierProvider*. URL: [https://riverpod.dev/docs/providers/state\\_notifier\\_provider](https://riverpod.dev/docs/providers/state_notifier_provider) (cit. alle pp. 39, 82).
- [64] *Swagger*. URL: <https://swagger.io/> (cit. a p. 42).
- [65] *Switch class*. URL: <https://api.flutter.dev/flutter/material/Switch-class.html> (cit. a p. 46).
- [66] *Text class*. URL: <https://api.flutter.dev/flutter/widgets/Text-class.html> (cit. alle pp. 44, 46, 52–54, 56).
- [67] *TextButton class*. URL: <https://api.flutter.dev/flutter/material/TextButton-class.html> (cit. alle pp. 47, 67, 80).
- [68] *TextEditingController class*. URL: <https://api.flutter.dev/flutter/widgets/TextEditingController-class.html> (cit. alle pp. 54, 55, 70).
- [69] *TextField class*. URL: <https://api.flutter.dev/flutter/material/TextField-class.html> (cit. a p. 55).
- [70] *TextFormField class*. URL: <https://api.flutter.dev/flutter/material/TextFormField-class.html> (cit. a p. 48).
- [71] *TextTheme class*. URL: <https://api.flutter.dev/flutter/material/TextTheme-class.html> (cit. a p. 81).
- [72] *Theme class*. URL: <https://api.flutter.dev/flutter/material/Theme-class.html> (cit. a p. 79).
- [73] *ThemeData class*. URL: <https://api.flutter.dev/flutter/material/ThemeData-class.html> (cit. alle pp. 79, 81).
- [74] *ToggleButtons class*. URL: <https://api.flutter.dev/flutter/material/ToggleButtons-class.html> (cit. a p. 56).
- [75] *Wikipedia*. URL: [https://en.wikipedia.org/wiki/Main\\_Page](https://en.wikipedia.org/wiki/Main_Page) (cit. a p. 92).
- [76] *Xcode*. URL: <https://developer.apple.com/xcode/> (cit. a p. 42).