UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Tesi di laurea

PROGETTAZIONE ED IMPLEMENTAZIONE DI UN'APPLICAZIONE MOBILE PER SUPPORTARE IL COORDINAMENTO DI ATTIVITÀ LAVORATIVE IN AMBITO SANITARIO

Relatore:
Prof. Emanuele Di Buccio

Laureando:
Leonardo De Marchi

ANNO ACCADEMICO 2023-2024

DATA DI LAUREA 27/09/2024

Sommario

Il presente documento descrive l’attività di tirocinio svolta presso INFOC.E.R.SRL, software house nel settore della sanità pubblica.
Il periodo lavorativo ha previsto la creazione di un’applicazione mobile che agevoli lo svolgimento del lavoro di un’equipe di ispettori sanitari, fornendo loro un’agenda e la possibilità di integrazione con il servizio Google Calendar.
Il prodotto si basa su SIRe (Sistema Informativo Regionale), ed è stato denominato SIRe App. SIRe consiste in una piattaforma web in corso di sviluppo da parte dell’azienda, in cui gli ispettori sanitari locali potranno gestire i propri impegni lavorativi ed inviare/consultare referti. SIRe App ne costituisce l’unica parte mobile nativa.
Vengono presentate la progettazione e l'implementazione dell’app tramite l’utilizzo del framework Flutter ed il linguaggio di programmazione Dart.

Ringraziamenti

Desidero ringraziare il mio relatore per la sua disponibilità, la mia famiglia e gli amici per il loro sostegno.

Padova, Settembre 2024 Leonardo De Marchi

Indice

Capitolo 1

Introduzione

All'interno del documento, per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: parola$_g$

Nel seguente capitolo viene riportata una descrizione dell'azienda presso cui è stata svolta l'attività di tirocinio e l'analisi degli obiettivi preposti all'inizio del percorso.

1.1 Contesto aziendale

Info.C.E.R. s.r.l. è una software house che progetta, produce e realizza una vasta gamma di soluzioni software, in particolare per il settore della sanità pubblica, sia umana in tutte le sue dislocazioni -distretti, laboratori, fondazioni, *AULSS*$_g$- che veterinaria, come l'istituto Zooprofilattico Sperimentale delle Venezie.

L'azienda nasce nel 1996, vantando ad oggi un'esperienza quasi trentennale nel settore, ed attualmente opera in due sedi distaccate, una a Mestre nel veneziano ed una a Borgomanero in provincia di Novara.

Figura 1.1: logo dell'azienda

La collaborazione tra l'azienda e l'Università di Padova è divenuta particolarmente stretta in tempi recenti attraverso la realizzazione di una piattaforma di aiuti per la didattica, messa a disposizione dall'Università per sostenere la comunità studentesca e le famiglie durante il periodo di pandemia di SARS-CoV-2.
Gli aiuti comprendevano una SIM dati per internet, contributi di acquisto per notebook, abbonamenti ai mezzi pubblici e affitti ma anche merchandising siglato Unipd.

La sede in cui ho svolto la totalità delle ore di stage è quella di Mestre, all'interno della quale ho trovato un ambiente di lavoro accogliente e personale disponibile con cui si è instaurato un rapporto lavorativamente costruttivo ed allo stesso tempo di amicizia, mantenendo anche continui contatti con la sede piemontese.

1.2 Obiettivo del tirocinio

All'interno della pagina di presentazione aziendale, un estratto rilevante all'attività di tirocinio svolta è il seguente:

"Dalla fondazione, siamo stati orientati a fornire soluzioni e applicativi basati sul web e, negli ultimi anni, abbiamo rivolto un'attenzione particolare ad integrare tali soluzioni per i dispositivi mobili"

tratto da infocer.com/lazienda-la-nostra-storia-oggi

È proprio sulle soluzioni per dispositivi mobili che io ed un collega, anch'egli tirocinante, abbiamo focalizzato la nostra attività, che è consistita nella progettazione e nella realizzazione di SIRe App.

SIRe App rientra in un macro-progetto denominato SIRe (Sistema Informativo Regionale), piattaforma web in cui gli ispettori sanitari delle *AULSS* locali potranno gestire i propri impegni lavorativi ed inviare/consultare i referti.
SIRe verrà approfondito all'interno della sezione 2.1.

L'applicazione mobile fornisce ai suddetti utilizzatori un'agenda per facilitare l'organizzazione del lavoro e la possibilità di sincronizzazione delle prestazioni con il servizio *Google Calendar*$_g$, il tutto a portata di smartphone.

I requisiti iniziali dell'applicazione sono stati redatti dopo un periodo iniziale di colloqui con il committente, ed è stato deciso che una prima versione del progetto, considerati il numero di tirocinanti (2) ed il numero di ore dedicate (225), dovesse contenere i seguenti punti:

- autenticazione nel portale SIRe
- visualizzazione eventi dell'agenda SIRe
- schermata riassuntiva utente SIRe
- integrazione con *Google Calendar*
- visualizzazione notifiche ricevute su SIRe

Capitolo 2

Il progetto

Nel seguente capitolo viene presentato il progetto a partire dalla sua ideazione, per poi analizzare l'ambiente ed il contesto in cui verrà inserito fino ad arrivare alle specifiche ed alle tecnologie utilizzate.

2.1 Contesto generale e motivazioni del progetto

SIRe, o Sistema Informativo Regionale, nasce come piattaforma web destinata a professionisti della sanità pubblica, in particolare veterinaria. Permette ai responsabili di zona di gestire lo svolgimento delle mansioni lavorative di un'equipe di ispettori sanitari, impegnati nei controlli di qualità e sicurezza alimentare nei luoghi adibiti ad allevamento, macello e distribuzione di prodotti animali.
Gli utenti riceveranno su SIRe informazioni sulle ispezioni da effettuare, sotto forma di avviso su agenda con le specifiche del luogo, data e modalità. Successivamente potranno caricare sulla piattaforma i referti e ricevere ulteriori indicazioni a riguardo.

Ci sono due modalità di organizzazione degli utenti all'interno della piattaforma, una gerarchica ed una per privilegi.

La prima consiste nel fornire accessi e potere esecutivo maggiori man mano che si sale la scala gerarchica, individuando **gruppi** propri dell'organigramma interno alle aziende sanitarie locali.

Ad esempio:

- la regione vede e coordina le varie *AULSS* (9 in totale nella regione Veneto);

- all'interno di ogni *AULSS*, il responsabile vede e coordina sia il reparto di igiene alimenti che veterinario;

    → in alcune aziende sanitarie può essere richiesta separazione tra i responsabili dei due reparti. In casi come questo si limita l'assegnazione del gruppo base (ad es. igiene alimenti) senza permettere accessi indesiderati;

- i gruppi possono essere assegnati anche con valenza geografica, in base alle zone territoriali desiderate. In questo modo la gestione dei gruppi è gerarchica ma anche trasversale, poiché è possibile avere un utente responsabile di zona su Treviso e su Verona allo stesso tempo.

La seconda modalità di organizzazione degli utenti è quella per privilegi, che prevede l'utilizzo dei **ruoli**.
A seconda del ruolo, si possiedono determinati privilegi (permessi) all'interno di SIRe, ad esempio la facoltà di aggiungere o modificare prestazioni ai dipendenti nel caso di un responsabile.
Tuttavia all'interno di SIRe App, la logica dei privilegi non viene applicata poiché le prestazioni vengono solamente visualizzate, senza la possibilità di modificarle, nonostante l'utente possieda i permessi necessari. Parte della schermata riassuntiva dell'utente sarà comunque dedicata al ruolo, per ottenere una panoramica più completa possibile.

Considerati i luoghi di lavoro in cui SIRe verrà utilizzato, si nota che spesso i computer disponibili sono in condivisione tra più colleghi, dovendo quindi prestare attenzione a mantenere privato il proprio account effettuando di continuo accesso e disconnessione.
SIRe App si presenta come una soluzione comoda, privata ed efficace affinché gli utenti possano visionare la propria agenda ed il proprio account all'interno dello smartphone.
Si apre così la possibilità di avere la piattaforma sempre con sé.

2.2 SIRe App

SIRe App è un'applicazione mobile che mira ad interfacciarsi con l'utente (ispettore sanitario delle *AULSS* locali) per fornirgli un'agenda lavorativa di supporto con la possibilità di sincronizzare quest'ultima con Google Calendar, al fine di gestire al meglio gli impegni lavorativi assieme a quelli personali.

Figura 2.1: logo dell'applicazione SIRe

In aggiunta ad una prima visione mensile dell'agenda, l'utente potrà selezionare il dettaglio di un singolo evento, sul quale sarà riportato l'indirizzo della prestazione da effettuare nonché le specifiche da seguire.

Sarà presente un riepilogo delle informazioni dell'account SIRe con cui si è effettuato l'accesso, dando all'utente la possibilità di cambiare **gruppo** (nel caso ne avesse associati più di uno) e aggiornando l'agenda di conseguenza.
Per la definizione di **gruppo** si può fare riferimento alla sezione 2.1.

Accompagna la visione d'insieme dell'account, la sezione notifiche; le notifiche vengono reperite dalla piattaforma SIRe e rese consultabili anche all'interno dell'applicazione.

Successivamente al login con il proprio account Google, l'utente avrà accesso alle preferenze di sincronizzazione, potendo scegliere di impostare l'intervallo di date e se considerare date passate o meno.

Figura 2.2: logo di Google Calendar

L'obiettivo della descrizione riportata in questa sezione era quello di fornire una presentazione del progetto nel suo insieme, una più dettagliata analisi dei requisiti e della progettazione e sviluppo sono presenti rispettivamente alla sezione 3 e 4.

2.3 Metodo di lavoro: Agile

Considerate le dimensioni contenute del progetto e le interazioni, anche su base giornaliera, con il cliente tramite tutor aziendale (nonché fondatore dell'azienda stessa), si è dimostrato necessario adottare un processo basato su rapidità di sviluppo e capacità di adattamento alle continue evoluzioni del contesto.

Il metodo *Agile* si è rivelato efficace, considerata la presenza di requisiti talvolta mutevoli ed il ritmo con cui venivano effettuati incrementi. Tenuto conto del periodo di tempo ristretto, considerando anche una breve formazione iniziale, per portare a termine un progetto di questo tipo dal principio, si stabilirono solamente gli incrementi iniziali, con i successivi a dipendere dallo stato di avanzamento del lavoro.

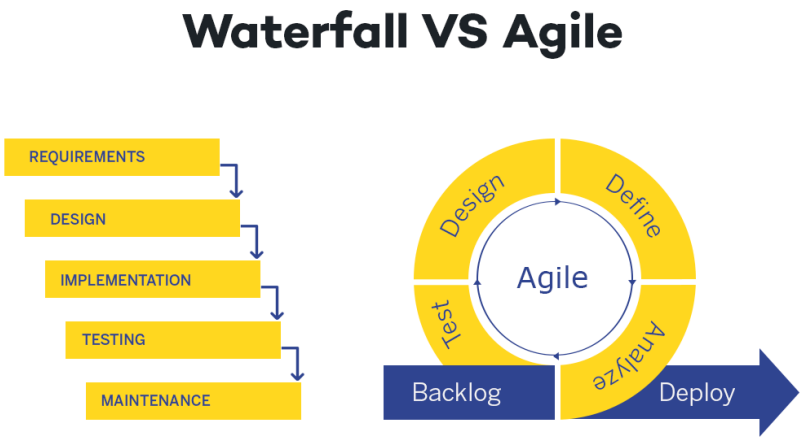

Figura 2.3: Rappresentazione grafica dei modelli *Waterfall* ed *Agile*

Come riportato in **Figura 2.3**, il modello a cascata (o *plan driven*) si compone del susseguirsi di fasi distinte (o stadi) che per iniziare devono attendere l'approvazione della fase precedente. Può rivelarsi inadatto per la produzione di software in quanto spesso non si tratta di un processo lineare e le fasi potrebbero sovrapporsi, ad esempio ottenendo nuove informazioni relative ad uno stadio precedente quando questo era già stato concluso. Il metodo *Waterfall* è tuttavia appropriato per grandi sistemi software, spesso sviluppati da più società, che necessitano di fondamenta solide e complete per operare autonomamente sui relativi sottosistemi.

Al contrario, l'approccio *Agile* è particolarmente utile per prodotti di piccole/medie dimensioni caratterizzati da un numero ristretto di *stakeholder*$_g$, con i quali è possibile instaurare un dialogo continuativo.
Agile nasce negli anni '90 e centra la sua filosofia attorno a 4 principi descritti nel Manifesto (*consultabile su agilemanifesto.org*), che danno maggiore importanza agli individui rispetto ai processi, al software funzionante rispetto alla documentazione esaustiva, alla

collaborazione con il cliente rispetto alla negoziazione dei contratti e alla capacità di rispondere al cambiamento rispetto al seguire un piano prestabilito.

Uno degli approcci più significativi di *Agile*, che racchiude la metodologia di processo utilizzata da me e dal collega con cui ho svolto l'attività di tirocinio, è l'*Extreme Programming (XP)*.
Le normali pratiche di sviluppo vengono spinte all'estremo, quindi si avranno, ad esempio, diverse versioni di un sistema da integrare in un solo giorno, mantenendo un ritmo (o *project heartbeat*) di sviluppo incrementale molto alto, caratterizzato da piccole ma continue *release*$_g$.

Figura 2.4: valori di *XP*

L'utilizzo di *XP* è stato facilitato da alcune *feature*$_g$ presenti in *Flutter* (vedere sezione 2.4), come la possibilità di effettuare *Hot Reload / Hot Restart*, che permettono di aggiornare il codice sorgente dell'app attualmente in esecuzione senza un riavvio completo e mantenendo lo stato (*Hot Reload)*.

Altre pratiche di XP applicate durante il periodo di tirocinio sono:
- integrazione continua dei task nel sistema
- cliente on-site: il tutor aziendale svolgeva a tutti gli effetti la funzione di rappresentante del cliente
- programmazione a coppie, in quanto eravamo due tirocinanti a lavorare a stretto contatto sullo stesso progetto
- proprietà collettiva: verifiche reciproche e possibilità di apportare modifiche al codice in ogni sua parte, senza avere aree riservate ad uno sviluppatore
- *refactoring*$_g$ continuo

2.4 Tecnologie di sviluppo

Viene qui riportata una breve descrizione dei componenti dell'ambiente di sviluppo, esplicitando le motivazioni e le caratteristiche per le quali sono stati scelti.

Flutter

Il primo componente da analizzare è il *framework*$_g$ all'interno del quale si è andati a costruire l'applicativo.
Flutter è un progetto *open-source*$_g$ di *Google* che ha come vantaggio principale la possibilità di creare applicazioni *mobile* multipiattaforma a partire da un unico codice sorgente, compilando nativamente sia in *iOS* che in *Android*. Ciò permette di sviluppare prodotti software fruibili da una gamma di dispositivi molto più ampia, potendosi concentrare su un unico codice.
La prima versione stabile del *framework* risale a Dicembre 2018, perciò si tratta di un prodotto recente ma in continua crescita, essendo sempre più richiesto come competenza ed avendo una vetrina importante come *Google*.

Figura 2.5: logo del *framework Flutter da flutter.dev*

Una peculiarità di Flutter, oltre alle *feature* di *Hot Reload* e *Hot Restart* già presentate, è la modalità con cui si progetta l'interfaccia utente: dato un generico componente grafico detto *Widget*, è possibile creare *Widget* più complessi assemblando e componendo i "tasselli" di base, fino ad arrivare alle più disparate schermate di *UI*$_g$.
La lista di tutti i *Widget* disponibili è consultabile sul sito della documentazione di *Flutter* (*Widget Catalog* su *flutter.dev*).

Altro punto cruciale nello sviluppo di applicazioni *mobile* è la gestione dello **stato**.

Con **stato** si intende l'insieme di dati che rappresentano la condizione dell'applicazione in un dato momento, e determina come l'app risponde alle azioni dell'utente e cosa viene visualizzato.
Di conseguenza, la gestione dello stato è il processo di monitoraggio, coordinamento ed aggiornamento dello stato stesso per garantire che ciò che viene elaborato e mostrato all'utente sia corretto, coerente e rispecchiato dalla *UI*.

In *Flutter*, le componenti grafiche si dividono in due famiglie, *Stateless Widget* e *Stateful Widget*, rispettivamente *Widget* privi di stato e aventi stato.

Di quest'ultimi, è possibile gestire lo stato delle singole componenti utilizzandone le proprietà dinamiche locali.
Per condividere e gestire lo stato tra i vari *Widget*, e quindi tra le varie schermate dell'applicazione, si utilizzano i *Provider*. Un *Provider* è un meccanismo di gestione dello stato che consente di esporre e condividere dati, in modo da renderli accessibili da ogni punto dell'app per poter coordinare correttamente il comportamento dinamico del software.

Infine la presenza di una straordinaria quantità di librerie importabili e comodamente utilizzabili rende il *framework* particolarmente appetibile per uno sviluppo incentrato sulle *main features* del proprio prodotto, risparmiando tempo e risorse altrimenti spesi per la gestione di compiti (*task*) minori ma comunque impegnativi.

Dart

Dart costituisce la base di *Flutter*, fornendo il linguaggio ed i *runtime*$_g$ su cui si basano le app di quest'ultimo.
Anche *Dart* è sviluppato da *Google*, nato con lo scopo di sostituire *JavaScript* come principale attore per lo sviluppo di applicazioni web. L'obiettivo di *Dart*, come riportato nella documentazione, è quello di offrire un linguaggio di programmazione più produttivo possibile per lo sviluppo multipiattaforma.

Figura 2.6: logo del linguaggio *Dart da* dart.dev

Il linguaggio è orientato agli oggetti, ed è *type safe*, utilizzando il controllo sul tipo per garantire che il valore di una variabile corrisponda sempre al tipo statico della stessa.
Inoltre la stesura del codice è flessibile, consentendo l'utilizzo di un tipo *dynamic* combinato con controlli *runtime*$_g$, che può rivelarsi utile durante prove o per codice che necessita di essere dinamico.
Infine il linguaggio pone particolare attenzione alla *null safety*: i valori non possono essere *null* a meno che questa possibilità non venga dichiarata esplicitamente.

Insomnia

Per svolgere la quasi totalità delle sue funzioni, SIRe App comunica con la struttura di *backend*$_g$ che l'azienda ha predisposto alle spalle della piattaforma SIRe.
La comunicazione avviene tramite un'architettura *API-RESTful* (*vedere sezione 4.1*), ed è stato possibile testare preventivamente le chiamate grazie ad *Insomnia*.

Figura 2.7: logo di *Insomnia da* insomnia.rest

Insomnia è un' applicazione desktop *open source* che semplifica l'interazione, la progettazione, il debug ed il test delle *API*. Dotato di semplicità di utilizzo e di una interfaccia utente intuitiva, ha aiutato a superare problemi di comunicazione tra l'applicazione ed i servizi di *backend* altrimenti nascosti e di difficile risoluzione.

2.5 Tecnologie di supporto

Segue una breve descrizione delle tecnologie adottate per affiancare lo sviluppo. Le prime due sono state scelte liberamente dai tirocinanti, mentre l'ultima è stata adottata perchè utilizzata dall'azienda ospitante.

GitHub

GitHub è un servizio di *hosting*$_g$ per progetti software, che permette agli sviluppatori di archiviare, gestire e condividere il proprio codice.
Il nome "*GitHub*" proviene dal fatto che consiste nell'implementazione dello strumento di controllo versione distribuito *Git*.

Figura 2.8: logo di *GitHub*

Figura 2.9: logo di *Git*

L'utilità dei sistemi di controllo di versione consiste nel mantenere traccia dei cambiamenti di un progetto software, consentendo al bisogno di tornare ad una specifica versione.
Negli anni ci sono state diverse generazioni di *VCS* (*Version Control System*). A partire dai sistemi locali, dove le differenze tra file (*patch*) venivano memorizzate su disco, passando per i sistemi centralizzati, per permettere la collaborazione tra più *team*, con un singolo *server* contenente tutte le versioni del file ed i *client* che effettuano il *checkout* dei file dal *server*. Alcuni punti critici, come il fatto che un *server* di quel tipo rappresenta un *single point of failure*, hanno portato allo sviluppo ed alla diffusione dei sistemi distribuiti, dove *Git* è il più utilizzato.
Git nasce nel 2005 come sistema gratuito ed *open source* per gestire lo sviluppo del Kernel Linux. La nuova concezione dei sistemi distribuiti permette ai *client* di scaricare una copia completa (*clone*) del *repository* -contenente ciò su cui si sta lavorando- in modo tale da avere un *backup* completo su ciascun *client*.

Figma

Per sperimentare le possibili soluzioni dell'interfaccia grafica, *Figma* si è rivelato un editor di grafica pratico ed affidabile, con la possibilità di lavorare in modo condiviso ed in tempo reale su uno stesso progetto da terminali diversi.

Figura 2.10: logo di *Figma*

Ottenuti i *mock up*$_g$, si è cercato di riprodurre i layout all'interno del codice *Flutter* nel modo più simile possibile.

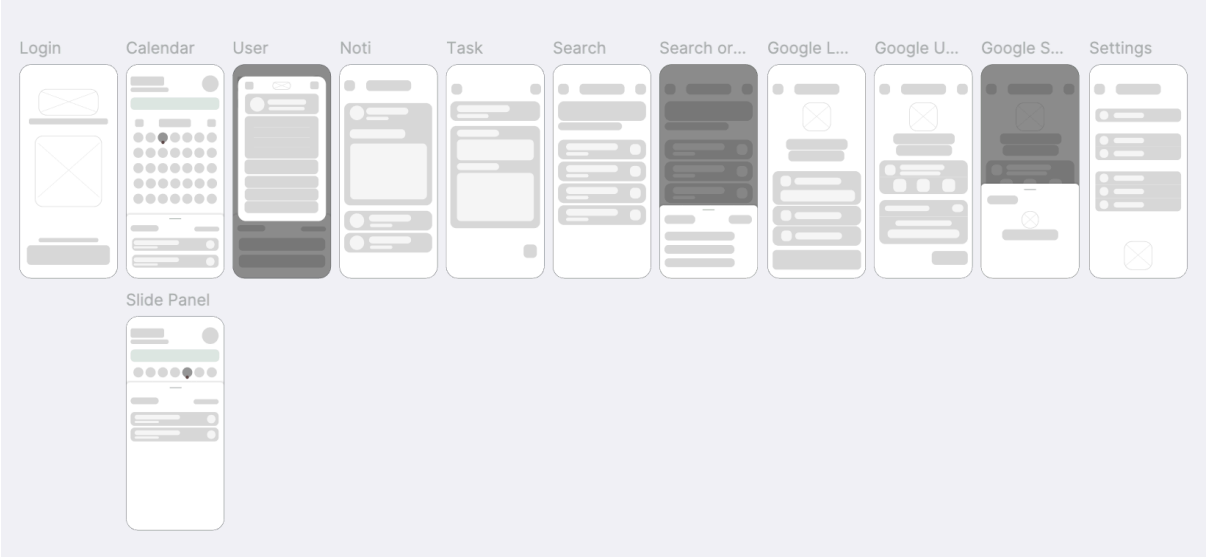

Figura 2.11: schermata di lavoro su *Figma*

Teams

Microsoft Teams, nota piattaforma di comunicazione che combina videoconferenze e condivisione di contenuti, è stata di centrale importanza per la comunicazione tra tutti i membri del team in presenza, in *smart working* ed operanti nella sede distaccata dell'azienda ospitante.

Figura 2.12: logo di *Microsoft Teams*

Capitolo 3

Analisi dei requisiti

Nel seguente capitolo viene presentato l'approccio adottato ed i risultati ottenuti durante la fase di analisi dei requisiti, condotta tra tirocinanti e l'azienda ospitante.

L'analisi dei requisiti consiste in una serie di attività dedite alla raccolta ed alla definizione dei servizi che il sistema deve fornire e dei vincoli operativi che deve rispettare.

La fase di raccolta si è sviluppata attraverso una serie di colloqui, principalmente con il committente o tutor aziendale, ma anche con alcuni colleghi per ottenere più informazioni possibili sul sistema da punti di vista differenti.
Il primo passo è stato essere introdotti a SIRe, per comprendere il contesto e gli obiettivi aziendali relativi a questo progetto. Successivamente, è stato possibile comprendere le necessità del committente tramite interviste aperte, ed in un secondo momento ultimare la raccolta con domande mirate (interviste chiuse).
Si è giunti quindi ad una specifica dei requisiti mediante diagramma dei casi d'uso e notazione in linguaggio naturale strutturato.

3.1 Specifica

Il linguaggio *UML* (*Unified Modeling Language*) permette di analizzare, documentare e visualizzare un sistema *software* tramite l'utilizzo di modelli visuali.
Esistono diverse tipologie di diagrammi *UML*, come ad esempio i diagrammi di classe, diagrammi di sequenza o diagrammi di stato.

Per questa analisi dei requisiti si è preferito riportare il diagramma dei casi d'uso, poiché permette di visualizzare in modo semplice e diretto lo scopo del prodotto e le funzionalità richieste dal committente. Il diagramma è composto dagli attori, cioè le persone o i sistemi coinvolti in un'interazione, ed i casi d'uso che descrivono i requisiti funzionali in termini generali.

Essendo l'applicativo parte di un progetto più esteso, con *Utente* si fa riferimento agli utilizzatori già registrati nel portale SIRe.

In SIRe App non è possibile avviare il processo di registrazione, ma è possibile effettuare l'accesso al portale tramite *in App WebView*, un *widget* che integra un *browser web* all'interno di una schermata dell'applicazione, permettendo la fruizione di contenuti web senza uscire dall'app stessa.

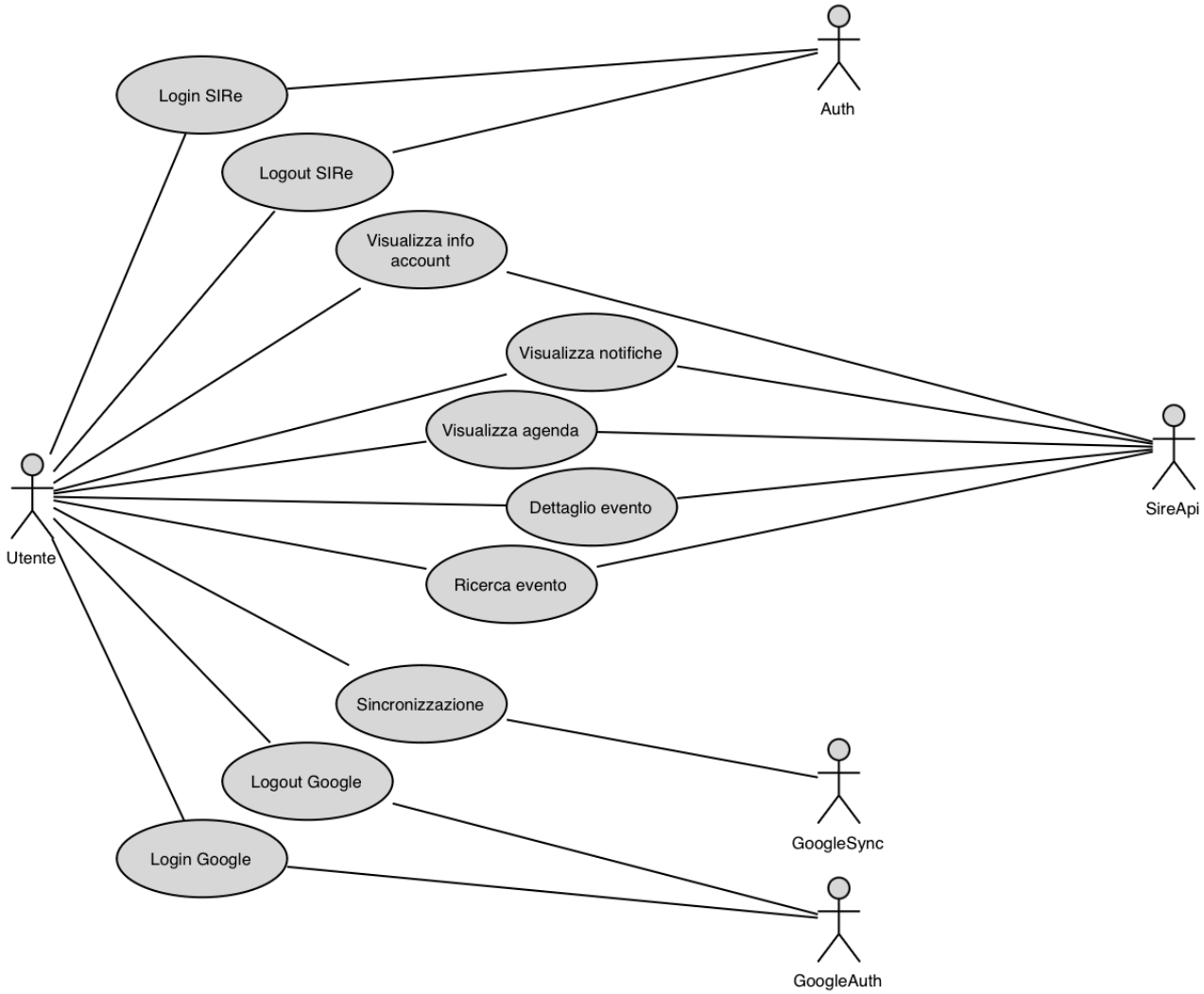

Figura 3.1: diagramma dei casi d'uso

La specifica in linguaggio naturale strutturato prevede l'utilizzo di schemi o tabelle per migliorare l'organizzazione e la leggibilità dei requisiti pur mantenendo l'espressività del linguaggio naturale.

In accordo con gli standard aziendali, i requisiti sono stati suddivisi in requisiti di business, tecnici funzionali e tecnici non funzionali.

3.2 Requisiti di business

I requisiti di business riguardano gli obiettivi strategici e le necessità che un progetto software deve soddisfare.
Solitamente prevale il punto di vista del committente, per definire cosa l'azienda (o chiunque abbia commissionato il progetto) vuole ottenere, lasciando il come alla prossima categoria di requisiti.

Codice	Descrizione
RB001	L'applicazione deve permettere di autenticarsi nel portale SIRe
RB002	L'applicazione deve permettere di visualizzare una schermata riassuntiva delle informazioni dell'account autenticato
RB003	L'applicazione deve permettere di visualizzare gli eventi dell'agenda SIRe
RB004	L'applicazione deve permettere di visualizzare il dettaglio di un evento dell'agenda SIRe
RB005	L'applicazione deve permettere di cercare un evento all'interno del calendario
RB006	L'applicazione deve permettere di visualizzare le notifiche ricevute su SIRe
RB007	L'applicazione deve permettere di autenticarsi con il proprio account Google
RB008	L'applicazione deve permettere di sincronizzare gli eventi di SIRe con il calendario personale di Google
RB009	L'applicazione deve permettere di visualizzare una schermata riassuntiva delle informazioni dell'account Google autenticato

Tabella 3.1: requisiti di business

3.3 Requisiti tecnici funzionali

I requisiti tecnici funzionali riguardano le funzionalità che l'applicazione deve avere per soddisfare i requisiti di business, specificando come il software deve essere costruito.
Il codice RSF indica Requisiti Specifici Funzionali.

Codice	Descrizione
RSF001	L'applicazione deve permettere di autenticarsi tramite web view in app al portale web di autenticazione SIRe
RSF002	L'applicazione deve permettere di visualizzare gli eventi dell'agenda SIRe riportati su un calendario mensile
RSF003	L'applicazione deve permettere di visualizzare il dettaglio di un evento dell'agenda SIRe
RSF004	L'applicazione deve permettere di visualizzare la posizione su mappa di un evento dell'agenda SIRe
RSF005	L'applicazione deve permettere di cercare un evento all'interno del calendario, specificando il nome dell'azienda in cui si svolgerà
RSF006	L'applicazione deve permettere di visualizzare l'elenco degli eventi corrispondenti alla ricerca ordinati in base alla data di svolgimento
RSF007	L'applicazione deve permettere di visualizzare l'elenco degli eventi corrispondenti alla ricerca ordinati in base al nome dell'evento
RSF008	L'applicazione deve permettere di visualizzare l'elenco degli eventi corrispondenti alla ricerca ordinati in base alla durata dell'evento
RSF009	L'applicazione deve permettere di visualizzare l'elenco degli eventi corrispondenti alla ricerca in ordine ascendente o discendente
RSF010	L'applicazione deve permettere di visualizzare una schermata di riepilogo delle informazioni dell'account SIRe con cui si è effettuata l'autenticazione
RSF011	L'applicazione deve permettere di visualizzare le notifiche ricevute su SIRe, distinguendole tra lette e non lette
RSF012	L'applicazione deve permettere di segnalare la presenza di notifiche non lette
RSF013	L'applicazione deve permettere di impostare le preferenze del tema
RSF014	L'applicazione deve permettere di autenticarsi con il proprio account Google
RSF015	L'applicazione deve permettere di sincronizzare gli eventi di SIRe con il calendario personale di Google, dato un intervallo di sincronizzazione di default

RSF016	L'applicazione deve permettere di sincronizzare gli eventi di SIRe con il calendario personale di Google, dato un intervallo di sincronizzazione scelto dall'utente
RSF017	L'applicazione deve permettere di visualizzare il resoconto delle sincronizzazioni legate ad un account

Tabella 3.2: requisiti tecnici funzionali

3.4 Requisiti tecnici non funzionali

I requisiti tecnici non funzionali riguardano gli attributi qualitativi di un sistema software, piuttosto che le funzionalità specifiche. Stabiliscono i criteri secondo i quali il sistema è considerato adeguato, in termini ad esempio di sicurezza o usabilità.

Codice	Descrizione
RSNF001	L'applicazione deve essere utilizzabile sul sistema operativo Android a partire dalla versione 6
RSNF002	L'applicazione deve essere utilizzabile sul sistema operativo iOS a partire dalla versione 13
RSNF003	L'autenticazione all'applicazione deve essere fatta tramite credenziali utilizzate sul portale SIRe
RSNF004	Il framework di sviluppo deve essere Flutter
RSNF005	Il programma deve rispettare i requisiti disposti dal codice in materia di protezione dei dati personali (by design)

Tabella 3.3: requisiti tecnici non funzionali

Capitolo 4

Progettazione e sviluppo

Nel seguente capitolo viene analizzata la struttura implementativa del progetto, partendo dall'architettura fino ad arrivare ai design pattern utilizzati all'interno del codice, soffermandosi sulle parti interamente progettate e sviluppate dal sottoscritto.

4.1 Architettura

SIRe, come già riportato in precedenza, è un progetto molto esteso di cui SIRe App costituisce l'unica parte *mobile* nativa. Il *backend* dunque è interamente opera di Info.C.E.R. s.r.l. e si interfaccia con l'app tramite un'architettura *API-RESTful*.
Le *API*, acronimo di *Application Programming Interface*, sono un insieme di definizioni e protocolli che permettono a due componenti *software* di comunicare tra loro. Definiscono i metodi ed i dati della chiamata e della successiva risposta, consentendo di integrare funzionalità all'interno delle applicazioni in modo standardizzato e iterabile.
All'interno delle *API* è possibile implementare i principi *REST*, o *Representational State Transfer.* Si tratta di un insieme di linee guida architetturali che l'*API* deve seguire per essere considerata *RESTful*, ad esempio avere un'architettura *client-server* con richieste gestite tramite *HTTP*. Per quanto riguarda *SIRe,* l'informazione viene consegnata nel formato *JSON* (*JavaScript Object Notation*), che rappresenta un formato di interscambio dati indipendente dal linguaggio e facilmente leggibile da persone e macchine.

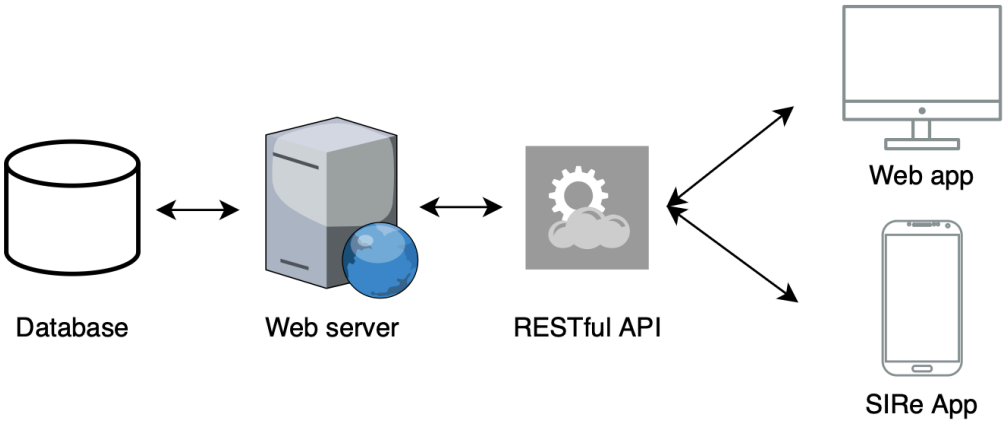

Figura 4.1: rappresentazione grafica dell'architettura utilizzata

Le *API* sono sviluppate in *Java*, mentre per il lato *frontend*$_g$ della *web app* l'azienda ha deciso di utilizzare *ext JS*, una libreria JavaScript adatta allo scopo.

SIRe App si basa su un'architettura interna sviluppata su 3 *layers* (livelli), per permettere una discreta separazione delle responsabilità, facilitando la manutenzione e l'estensibilità del codice.
La struttura del progetto verrà analizzata successivamente, ma si può far notare come all'interno delle varie *features* sia evidente la suddivisione in *layers*.

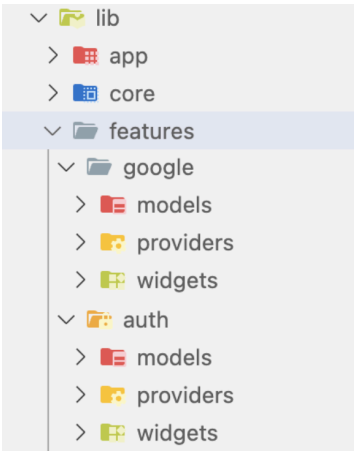

Figura 4.2: struttura cartelle illustrativa dei 3 *layers*

I livelli sono i seguenti:

Presentazione (*Presentation Layer*)

- **Descrizione:** Questo livello si occupa dell'interfaccia utente. Al suo interno vengono gestite la visualizzazione dei dati e le interazioni con l'utente.

- **Implementazione:** La cartella '*widgets*' rappresenta questo *layer*. Contiene le schermate dell'applicazione formate appunto dai *widgets* di *Flutter*, che consentono di modellare la *UI* e di gestire le interazioni con l'utente.
  Il *presentation layer* richiama i *provider* per ottenere i dati, presentandoli all'utente attraverso i *widgets*.

Logica e gestione dati (*Domain Layer*)

- **Descrizione:** Questo livello è responsabile della logica di business e della gestione dei dati. Qui vengono definiti i *provider*, presentati alla sezione 2.4, che agiscono da intermediari tra il livello di presentazione ed il livello dei modelli di dati, elaborando i dati ma implementando anche logiche specifiche come la gestione degli errori.

- **Implementazione:** La cartella '*providers*' racchiude il *domain layer*. Qui si trova il codice che elabora i dati dei modelli e comunica con l'interfaccia utente. All'interno del progetto si è scelto di utilizzare *Riverpod* per la gestione dello stato.
  Riverpod è un'evoluzione del pacchetto base '*provider*' interno a *Flutter*, che apporta migliorie come la possibilità di creare ed utilizzare *provider* autonomi indipendenti dal *widget tree*, ossia la struttura gerarchica di *widget* che va a comporre la *UI*.

Accesso ai dati (*Data Layer*)

- **Descrizione:** Questo livello si occupa della rappresentazione e della serializzazione o deserializzazione dei dati. Viene definita la struttura dei dati che verranno poi gestiti internamente all'applicazione.

- **Implementazione:** La cartella '*models*' contiene la struttura dei dati e la logica di serializzazione, per ottenere un formato gestibile agevolmente. Si è scelto di utilizzare la libreria *Freezed*, con lo scopo di facilitare la creazione di modelli di dati robusti in *Flutter*. Supporta la serializzazione e deserializzazione *JSON* e viene utilizzata per creare classi immutabili per definizione, con la possibilità di creare copie modificabili dell'oggetto originale attraverso il metodo *copyWith* fornito dalla libreria.

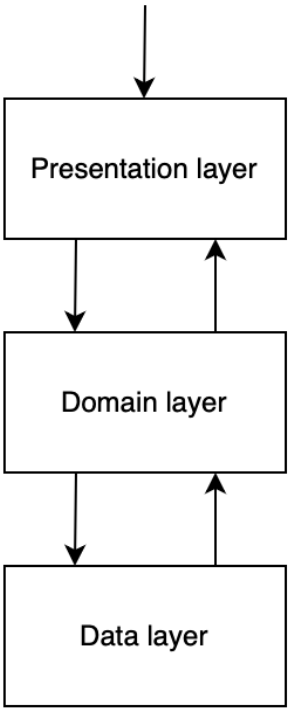

Figura 4.3: l'interazione tra i *layers* è bidirezionale

4.2 Struttura del progetto

Per analizzare l'applicazione in modo completo è bene descriverne la struttura, che risulta anche esplicativa delle funzionalità presenti.

All'interno di *Flutter* tutto il codice *Dart* è contenuto nella cartella \lib, che contiene:

- *main.dart*: *entry point* dell'applicazione. Inizializza le funzionalità di base e avvia App.
- suddivisione in sottocartelle.

La prima suddivisione consiste in:

\app
Contiene *app.dart*, bootstrap dell'applicazione che ne inizializza il tema ed il *router.*

\core
Sono racchiuse le componenti alla base dell'applicazione, risorse e funzionalità utilizzate in modo trasversale all'interno del progetto.

- **\config:** vengono definiti i temi e le impostazioni di configurazione di *Firebase,* piattaforma *Google* utilizzata per poter effettuare il login con account *Google*.
- **\constants:** le costanti utilizzate nel progetto, racchiuse in un unico posto per migliorare la leggibilità e l'ordine.
- **\extensions:** contiene estensioni per classi esistenti, che aggiungono funzionalità alle versioni iniziali delle classi utilizzate.
- **\utils:** le *utilities,* cioè generiche funzioni o classi *helper* utilizzate in più parti del progetto, riducendo così la duplicazione del codice e favorendo la modularità. *Utilities* come la formattazione delle date e del tempo, o l'utilizzo di un *Debouncer*, classe che posticipa l'esecuzione di un'azione riducendo il numero di volte che viene eseguita (utile per non sovraccaricare le *API* di richieste).

\routing
Racchiude ciò che concerne *GoRouter*, pacchetto di *routing* per *Flutter*.
Un *router* è un componente che gestisce la navigazione tra le schermate, definendo un insieme di percorsi (*routes*). I percorsi mappano i nomi delle rotte, definite come stringhe *URL* (*Uniform Resource Locator*), alle pagine corrispondenti.
Un aspetto importante del *router* è la possibilità di integrarlo con il sistema di gestione dello stato dell'applicazione, coordinando la navigazione in base allo stato corrente. Ad esempio viene controllato se l'utente è autenticato, indirizzandolo alla *home page*, o in caso contrario alla schermata di login.

\features
Contiene la logica e le schermate delle funzionalità dell'applicazione.
In ogni *feature* è presente l'architettura a 3 *layer* come riportato nella sezione 4.1.
La suddivisione per *features* è la seguente:

- **\api:** gestisce le interazioni con le *API* per ottenere risorse come le prestazioni da inserire nell'agenda, le notifiche SIRe o verifica la presenza di un calendario *Google* associato all'utente, altrimenti ne inserisce uno nuovo.

- **\auth:** contiene la logica e le schermate relative all'autenticazione degli utenti. Si occupa di effettuare il login, salva i *token* di accesso all'interno del *FlutterSecureStorage*, ascolta i cambi di sessione e si autentica automaticamente se c'è una sessione valida, senza chiedere all'utente di rifare l'accesso.
  FlutterSecureStorage è un pacchetto per *Flutter* che garantisce l'archiviazione sicura di dati sensibili, come ad esempio i *token* di accesso, crittografando i dati e salvandoli all'interno del *local storage* dei dispositivi (*Keychain* su *iOS* e *Keystore* su *Android*).

- **\calendar:** gestisce le funzionalità del calendario. Il calendario visualizza le prestazioni in base al giorno selezionato e permette di aprirne i dettagli, tramite pannello espandibile nel lato inferiore della schermata.

- **\google:** include l'integrazione con i servizi *Google* quali *Google Sign In* e *Google Calendar*.
  A seconda se l'utente ha effettuato l'accesso *Google* o meno, viene visualizzata la schermata *Google Auth* o la schermata *Google Sync*, con il resoconto delle interazioni precedenti e la possibilità di sincronizzare i calendari impostando l'intervallo di date desiderato.
  Per evitare la sovrapposizione di eventi o la creazione di calendari duplicati all'interno di *Google Calendar*, l'oggetto *Calendar* viene legato al gruppo (sezione 2.1) correntemente selezionato dall'utente.
  Per l'invio degli eventi tramite *API* di *Google*, viene creata una richiesta $batch_g$, limitando la possibilità di raggiungere i *quote limits* anche in caso di molte interazioni da parte dell'utente. Per *quote limits* si intendono i limiti imposti da *Google* per regolare quante richieste un'app o un utente può effettuare ad una *API* in un determinato intervallo di tempo.

- **\home:** si occupa di fornire le prestazioni al calendario, ottenute dall'agenda SIRe, completando la *home page* del calendario con la sezione notifiche ed un *banner* per la sincronizzazione *Google*.

- **\notifications:** contiene *provider* e *widgets* responsabili della gestione e visualizzazione delle notifiche SIRe all'interno di SIRe App. È possibile segnare una notifica come letta o non letta, e la selezione sarà visibile su tutti i dispositivi.

- **search:** gestisce la funzionalità della ricerca di prestazioni all'interno del calendario SIRe. È possibile effettuare la ricerca fornendo una *query*$_g$ testuale che confronta i nomi delle prestazioni e un *range* di date, altrimenti impostate di default.
  I risultati sono ordinabili in base al nome, alla data o alla durata (ogni prestazione ha una durata associata), con ordine ascendente o discendente.
  Nel campo della *query* testuale è stata utilizzata un'istanza di *Debouncer* (definito in *\core\utils*), evitando di sovraccaricare il *server* di richieste.

- **settings:** racchiude la logica e le schermate per le impostazioni, comprendenti la scelta del tema, le informazioni sull'applicazione e le informazioni legali (*privacy policy* e termini di servizio).

- **shared:** repository comune di *widgets*, modelli di dati e servizi utilizzati da più *features*, come l'*Enum*$_g$ dei metodi *HTTP* supportati o gli *header*/*footer*$_g$ per le liste.

- **storage:** si trova il *provider Riverpod* per la gestione del *FlutterSecureStorage*.

- **task:** contiene la gestione delle singole prestazioni, estraendone il contenuto e stampandolo a video come *HtmlWidget*. È possibile condividere la prestazione oppure, se l'indirizzo è disponibile, vengono mostrate le mappe disponibili e viene cercato l'indirizzo sulla mappa selezionata.

- **user:** comprende il funzionamento della schermata riassuntiva dell'utente SIRe. Mostra le informazioni dell'utente, i ruoli ed i gruppi a cui appartiene. Consente di selezionare il gruppo, andando a modificare di conseguenza le prestazioni visibili.

4.3 Design Pattern

I *pattern*, nella programmazione orientata agli oggetti, comprendono la definizione di un problema ricorrente incontrato durante lo sviluppo software e la sua soluzione generale, applicabile a nuovi contesti. Catalogando e serializzando coppie di problemi/soluzioni, è possibile creare *template* utilizzabili da altri come elemento costitutivo di base.

All'interno del codice *Dart* sono stati utilizzati *design pattern* (*pattern* di progettazione) per migliorare la qualità e la manutenibilità del codice, oltre ad ottenere una più chiara separazione delle responsabilità per non creare eccessive dipendenze.

Utilizzo di *Riverpod* (*Singleton* e *Observer*)

La scelta implementativa di utilizzare *Riverpod* come *provider* ha permesso di rendere migliore anche la progettazione del codice, soddisfando alcune tipologie di *pattern*.

È possibile istanziare oggetti *Riverpod* con la proprietà *keepAlive* impostata a *true*. Questo assicura che una classe abbia una sola istanza, e che quell'istanza sia globalmente accessibile emulando il comportamento del *pattern Singleton* e favorendo la modularità del codice.

Inoltre i *provider* sono definiti proprio per reagire ai cambiamenti dello stato, notificando qualunque oggetto sia in ascolto e permettendo di intraprendere azioni in base ai dati ottenuti dal *provider* stesso. Questo può ricordare il *pattern Observer*, *pattern* comportamentale cioè caratterizzante dei modi in cui classi od oggetti interagiscono.

Utilizzo di *Freezed (Factory* e *Immutable Object)*

Il *Factory pattern* è un *design pattern* creazionale che delega la creazione di un oggetto ad una sottoclasse o ad un metodo specifico, per non averla nel costruttore della classe principale.

All'interno del codice, i modelli di dati sono definiti utilizzando il pacchetto *Freezed*. Questo implementa costruttori definiti come *factory*, che delegano la creazione dell'oggetto ad una sottoclasse privata creata automaticamente da *Freezed*.
Inoltre, un altro esempio è dato dal metodo *factory fromJson*, la cui implementazione è generata automaticamente da *Freezed* e permette di trasformare un oggetto *JSON* in un'istanza della classe a cui è applicato.

Infine, come già presentato in precedenza, il pacchetto utilizzato permette di creare oggetti immutabili, escludendo il rischio di modifiche non intenzionali.

Builder

Si tratta di un *pattern* creazionale che permette di produrre diverse rappresentazioni di un oggetto a partire dallo stesso codice di costruzione. In *Flutter*, *widget standard* e *widget* più complessi vengono proprio costruiti tramite una funzione *build( )*, il che permette la composizione di oggetti della *UI* in modo flessibile e riutilizzabile.
Per *widget standard* si intendono quei *widget* forniti da *Flutter* come ad esempio *ListView*, mentre *widget* più complessi possono essere qualunque classe rappresentante una schermata dell'applicazione.

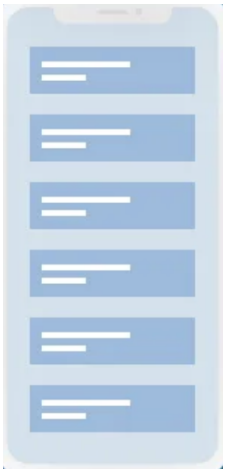

Figura 4.4: *mock-up* di una *ListView*

Lazy loading

Il *lazy loading* è un *pattern* considerato come tecnica di ottimizzazione, in quanto ritarda il caricamento di una risorsa fino a che essa non è realmente necessaria.
Ad esempio, un utente che esegue il comune '*scrolling*' all'interno di un *social*, vedrà le risorse della pagina caricarsi man mano che prosegue. Nonostante ci sia il rischio di incorrere in potenziali ritardi applicando questo *pattern*, si otterranno un tempo minore di caricamento iniziale, un uso più efficiente delle risorse ed in generale una migliore *User Experience*.
In *Flutter* le *ListView* applicano proprio questo principio, ed in SIRe App gli eventi sul calendario vengono ottenuti man mano che l'utente si sposta tra i vari mesi, evitando di fare un'unica grande richiesta all'avvio dell'app.

4.4 Principali responsabilità nel progetto

Come già accennato, durante il periodo di *stage* sono stato affiancato allo sviluppo di questo progetto da un collega, anch'egli tirocinante, con il quale ho condiviso sia le fasi di analisi e progettazione che le fasi implementative e di stesura della documentazione.
Nonostante la proprietà del codice fosse collettiva, con la possibilità da parte di ognuno degli sviluppatori di intervenire su tutte le sezioni del progetto, si possono individuare attività in cui uno dei due era ritenuto responsabile ed ha seguito maggiormente.
Andrò ora a presentare le parti che mi riguardano.

Calendario

Il calendario in SIRe App compone la maggior parte della schermata *home*, successivamente al login. Le necessità erano di avere una prima visione mensile, in modo che l'utente visualizzasse una panoramica degli impegni futuri, con una chiara segnalazione della presenza di eventi.
In base al numero di eventi presenti in un giorno, verranno visualizzati indicatori visivi, di numero pari agli eventi. Gli indicatori saranno rappresentati come punti, fino ad un massimo di quattro. Se il numero di eventi è superiore a quattro, verrà stampato come indicatore il carattere '+', per comunicare la presenza di ulteriori eventi non direttamente segnalati.

Un *widget SlidingUpPanel* riporta gli eventi del giorno selezionato ed, una volta espanso, focalizza la settimana scelta per uno spostamento più veloce tra giorni adiacenti.
Per poter cambiare mese o anno più rapidamente è stato implementato un selettore apposito, evitando di dover passare di mese in mese.

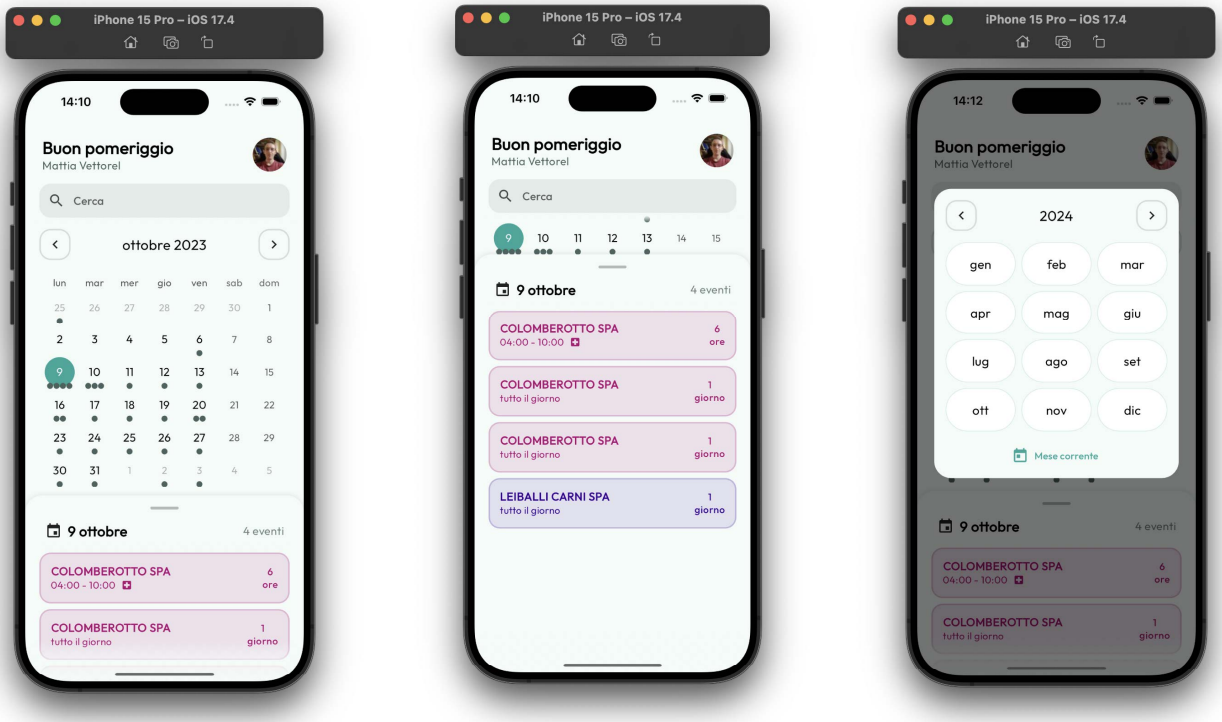

Figura 4.5: schermate calendario *SIRe App*

Visualizzazione eventi

L'evento, o prestazione, viene riportato all'interno dell'applicazione così come si trova sulla piattaforma SIRe. Il nome della ditta in cui effettuare l'ispezione è posto come titolo, mentre il corpo dell'evento comprende le specifiche operative che l'ispettore deve seguire.

Ad ogni nome viene associato un codice $hash_g$ e quindi un colore univoco, che sarà caratterizzante delle prestazioni con stesso titolo, come visibile in Figura 4.5.
È possibile ottenere le indicazioni stradali direttamente dall'evento, grazie al pacchetto *MapLauncher* che rende disponibili le mappe presenti sul dispositivo ed al pacchetto *GeoCoding* che traduce l'indirizzo della ditta in coordinate geografiche.

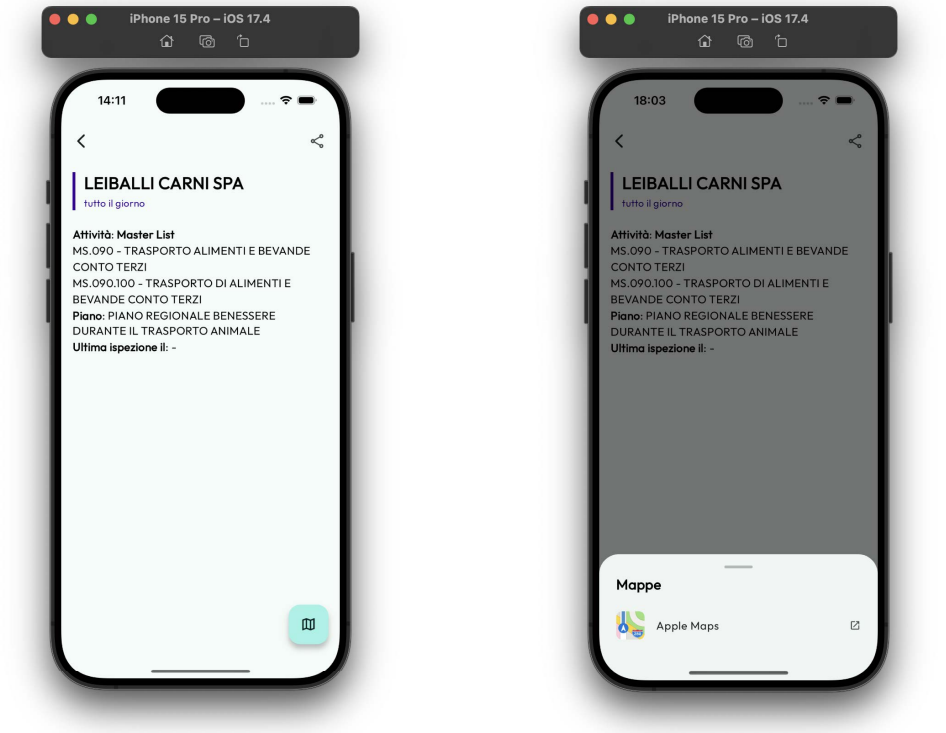

Figura 4.6: schermate visualizzazione eventi

Ricerca eventi

La ricerca degli eventi viene effettuata tramite titolo ed è consentita all'interno di un arco di tempo non superiore a sei mesi, per evitare di sovraccaricare il sistema.
I risultati della ricerca vengono stampati tramite *ListView* e viene evidenziato il titolo ed il giorno dell'evento. I risultati della ricerca sono selezionabili ed è possibile visualizzare il dettaglio dei singoli eventi.
I confronti nell'ordinamento possono essere effettuati per data, nome o durata a seconda della preferenza dell'utente. Inoltre i risultati possono essere stampati in ordine ascendente o discendente.

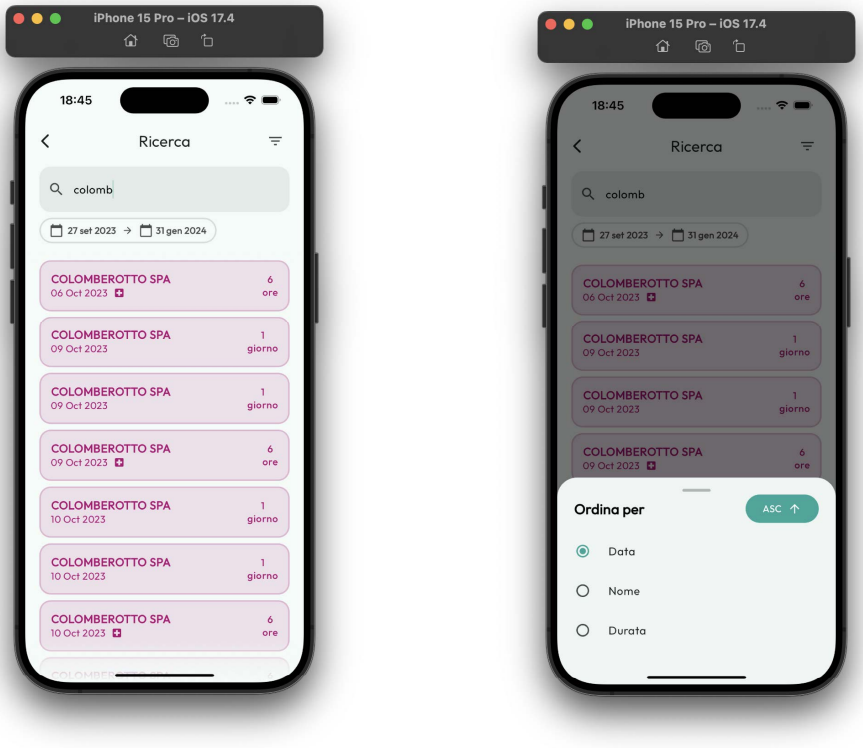

Figura 4.7: schermate ricerca eventi

Capitolo 5

Conclusioni

Nel seguente capitolo viene riepilogata l'esperienza di tirocinio, analizzando il grado di raggiungimento degli obiettivi e fornendo una valutazione personale del periodo lavorativo nel complesso.

L'obiettivo dell'esperienza di tirocinio era innanzitutto quello di introdurmi nel mondo del lavoro, potendo sperimentare una professione che mettesse alla prova le competenze da me acquisite fino ad ora durante il mio percorso universitario e di crescita individuale.
C'è stato un netto distacco dalla vita universitaria a cui ero abituato, poiché il periodo di lavoro si è sviluppato su sei settimane consecutive, con orario 9-18 dal lunedì al venerdì, permettendomi di vivere un lato della vita adulta a cui non mi ero ancora approcciato.
Questo ha portato ad un ampliamento notevole del mio bagaglio di esperienze, riuscendo anche a farmi osservare più da vicino un settore lavorativo che potrebbe essere parte della mia carriera e vita futura, consentendomi di sviluppare un'opinione più matura e solida a riguardo.

Dal lato del raggiungimento degli obiettivi formativi e delle competenze acquisite, posso ritenermi più che soddisfatto, opinione condivisa anche dall'azienda.
Ho iniziato questo percorso senza conoscere *Dart* e *Flutter*, ma grazie alle nozioni acquisite durante il corso di studi, un breve periodo iniziale di avvicinamento alle nuove tecnologie è stato sufficiente per raggiungere risultati soddisfacenti.
L'aver appreso tecnologie come *Dart* e *Flutter* ha reso il tirocinio un'esperienza estremamente utile, permettendomi di applicare le mie conoscenze pregresse, i miei metodi di ragionamento, di studio e di *problem solving* ad un contesto rapido ed esigente come il mondo del lavoro.
Ho avuto la fortuna di aver partecipato allo sviluppo di un progetto ambizioso e complesso all'interno dell'azienda, potendo allo stesso tempo creare un prodotto finito a partire dalla sua ideazione.
Questo mi ha permesso di seguire tutte le fasi di produzione, dall'analisi dei requisiti alla progettazione ed infine alla stesura, portandomi a conoscere nuove tecnologie di supporto che si sono rivelate necessarie per la loro praticità (*Figma*) o per la risoluzione di problemi (*Insomnia*).

Infine l'aver lavorato all'interno di un *team* mi ha permesso di crescere da un punto di vista umano e relazionale, imparando a conoscersi, aiutarsi e sostenersi tra colleghi con cui si condivide la maggior parte della giornata.

In conclusione mi ritengo soddisfatto del percorso svolto, e lo ritengo una valida scelta per completare il corso di studi in ingegneria informatica, sia dal lato formativo/professionale che sia dal lato umano/relazionale.

Glossario

AULSS Azienda Unità Sanitaria Locale Socio Sanitaria.

Backend parte di un sistema informatico che gestisce la logica, i processi, il database e l'infrastruttura senza interagire direttamente con l'utente finale.

Codice hash risultato di una funzione crittografica chiamata funzione di hash, che prende in ingresso un dato e lo trasforma in una stringa di lunghezza fissa.

Enum tipo di enumerazione, definito da un set di costanti.

Feature caratteristica, funzionalità.

Framework architettura logica di supporto sulla quale sviluppare un software.

Frontend parte dell'applicativo che gestisce l'interazione con l'utente.

Google Calendar sistema di calendari concepito da Google, integrato in ogni account Google.

Header/Footer Intestazione/Piè di pagina.

Hosting servizio che fornisce lo spazio e le risorse necessarie per archiviare file accessibili da utenti online.

Mock up modelli

Open source software non protetto da copyright e modificabile dagli utenti.

Query domanda o interrogazione per cercare elementi all'interno di un insieme di dati.

Refactoring ristrutturazione del codice volta a migliorarlo.

Release ciascuna nuova versione di un software.

Richiesta batch tipo di richiesta che permette di inviare più operazioni o comandi in un' unica transazione, riducendo il numero di richieste individuali da effettuare.

Runtime -inteso come ambiente di Runtime- è costituito dai componenti minimi necessari per creare ed eseguire un' applicazione.

Runtime indica il tempo in cui un programma viene eseguito.

Stakeholder ciascuno dei soggetti coinvolti in un progetto o nell'attività di un' azienda.

UI Interfaccia Utente.

Bibliografia

Riferimenti bibliografici

Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software.* 1995 Addison-Wesley Longman Publishing Co., Inc., USA

Sommerville I. *Ingegneria del Software* 2017 Pearson Italia, Milano-Torino, 10 edition

Chacon S., Straub B. *Pro git: Everything you need to know about Git.* 2014 Apress, second edition

Siti web consultati

Sito di InfoC.E.R. URL: https://www.infocer.com/.

Manifesto Agile. URL: https://agilemanifesto.org/.

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

Dart docs. URL: https://dart.dev/guides.

Insomnia docs. URL: https://docs.insomnia.rest/insomnia/get-started.

Freezed package. URL: https://pub.dev/packages/freezed.

Riverpod package. URL: https://riverpod.dev/docs/introduction/why_riverpod.