UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

DIPARTIMENTO DI INGEGNERIA ENZO FERRARI

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Realizzazione di un'applicazione Android per la valutazione dei clienti

Best Client

Tesi di laurea di:

Bianco Michael

Relatore:

Prof. Nicola Bicocchi

ANNO ACCADEMICO 2021/2022

Alla mia famiglia e ai miei amici

Indice

1. Introduzione

Le recensioni sono uno degli strumenti principali utilizzato soprattutto nel mondo online. Attraverso le recensioni una persona, ad esempio, posta davanti al problema dell'acquisto, magari con più possibilità di scelta, è in grado di orientarsi e scegliere l'alternativa più adatta a lei o che la fa sentire più sicura, oppure il caso tipico è quello di un servizio, dove le recensioni permettono di consultare in un attimo il parere di chi ne ha già usufruito.

1.1. Il potere delle recensioni

Uno studio svolto da Capterra mostra che il 90% degli italiani legge recensioni online prima di acquistare un prodotto o un servizio e che molti di loro si fidano più delle recensioni che delle raccomandazioni di amici. Tra le categorie più recensite c'è la ristorazione, uno dei settori in cui l'opinione di altri ha il potere di "attrarre" o "respingere" una persona che sta valutando se provare quel posto oppure sceglierne un altro. Nella valutazione c'è però da tenere a mente che il feedback che stiamo leggendo è un'opinione personale, pertanto è necessario analizzare anche le altre recensioni per farsi un'idea del prodotto/servizio che stiamo per comprare.

Figura 1. Il potere delle recensioni

Lo studio mostra anche le principali ragioni che portano una persona a scrivere un feedback sul prodotto che hanno acquistato, la prima è aiutare gli altri clienti ad orientarsi nella loro scelta, descrivendo quella che è stata la loro esperienza con l'impresa venditrice. La seconda motivazione è quella di aiutare l'azienda a migliorare il prodotto, suggerendo modifiche o evidenziando alcuni difetti che presenta un articolo in modo da correggere eventuali errori o aggiungere qualcosa in più. A seguire poi ci sono ragioni personali che spingono una persona a scrivere un commento, come il fatto di volere esprimere la propria soddisfazione o insoddisfazione su quel prodotto o servizio.

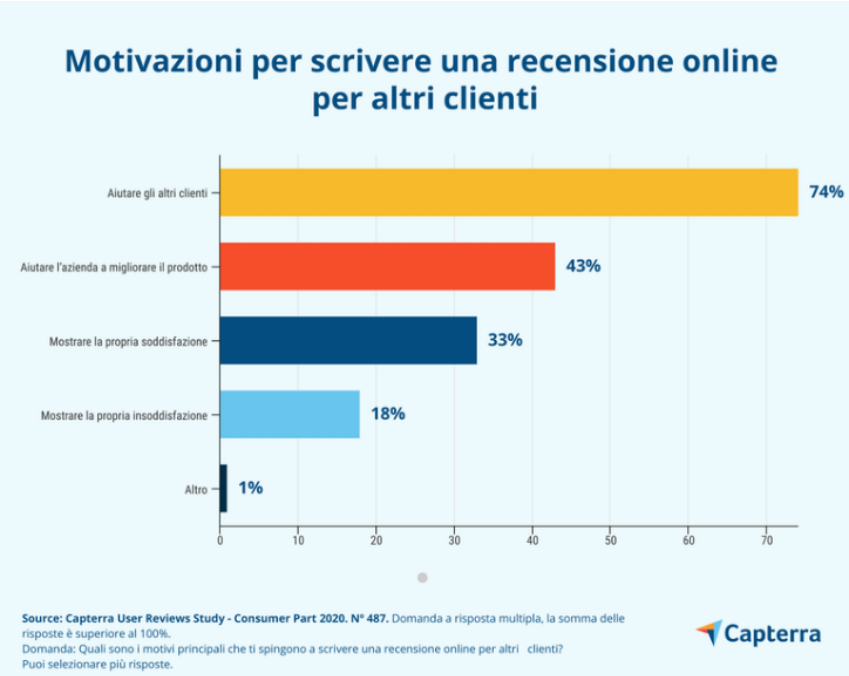

Figura 2: Le motivazioni delle recensioni

1.2. L'idea di BestClient

L'applicazione vuole dare la possibilità a chi possiede un'attività di recensire i propri clienti e dare un parere sull'interazione con essi, in modo che gli altri imprenditori possano avere un'idea del cliente una volta che questo utilizza uno dei loro servizi.

L'idea nasce da un ristoratore bolognese che, di fronte al problema di overbooking nei suoi locali, ha voluto un'applicazione che gli permettesse di scegliere i clienti in base alle sue valutazioni e a quelle degli altri ristoratori della zona che hanno già interagito con quel determinato cliente.

2. Struttura dell'applicazione

L'applicazione è composta da cinque pagine principali, di seguito le analizzeremo una ad una.

2.1. Schermata di Login

All'avvio dell'applicazione viene mostrata una schermata di login con username e password impostati nella fase di registrazione. Vengono inoltre richiesti i permessi di lettura del registro chiamate e dei contatti per utilizzare tutte le funzionalità dell'app.

L'utente non ancora iscritto può facilmente accedere alla pagina di registrazione cliccando sull'apposito link in basso.

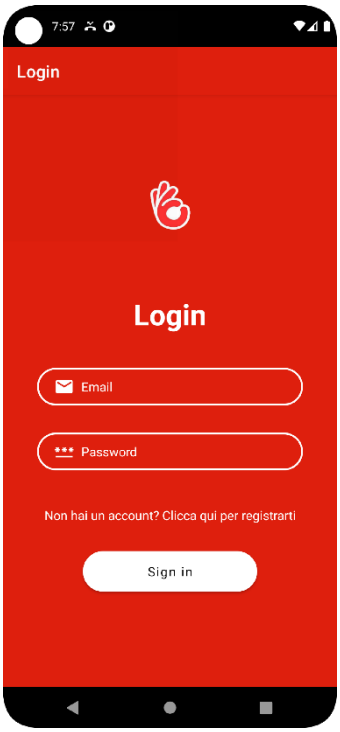

Figura 3: Login

2.2. Schermata di Registrazione

La schermata di registrazione permette di creare un account utilizzabile su tutti i dispositivi Android inserendo un'e-mail, una password e la partita IVA dell'attività per cui si sta utilizzando l'app. Attraverso il bottone "Registrati ed entra" viene inviata la richiesta di registrazione e si passa alla home page principale.

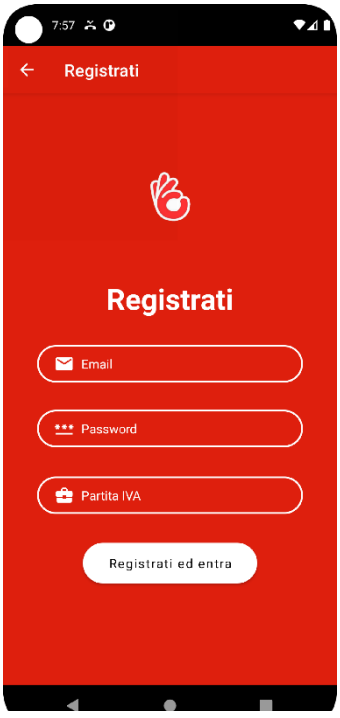

Figura 4: Registrazione

2.3. Home Page

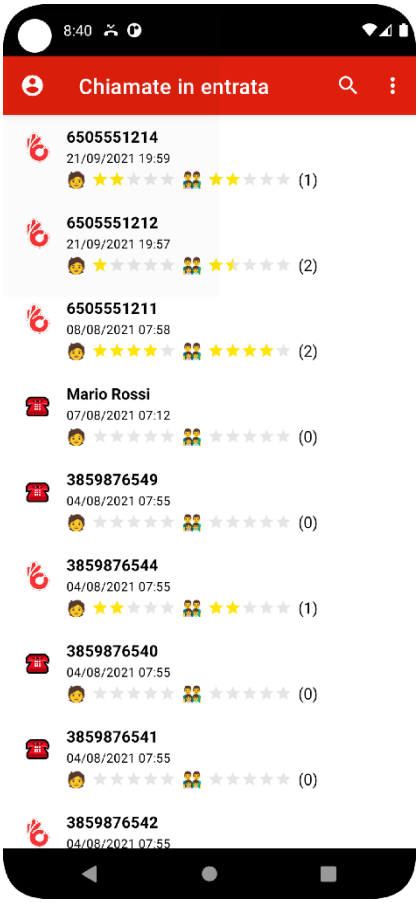

Figura 5: Home Page

La home page mostra le valutazioni assegnate ai vari numeri di telefono in base al filtro scelto, modificabile cliccando sui tre puntini in alto a destra. Si può filtrare in base alle chiamate in entrata, alle chiamate in uscita, alla rubrica contatti e alle valutazioni personali dell'utente stesso.

Nel caso in cui il numero valutato sia presente nella lista contatti viene visualizzato il nome del contatto per facilitare la lettura, altrimenti viene mostrato il solo numero telefonico.

Nella lista si possono notare quattro elementi principali per ogni numero:

- Il logo di BestClient per indicare che il numero è già presente all'interno del database in quanto valutato da almeno una persona, oppure il telefono per indicare che quel numero non è ancora stato valutato
- Il numero telefonico e la data dell'ultima chiamata
- La valutazione dell'utente e la valutazione collettiva di chi utilizza BestClient
- Il numero di valutazioni associate a quel numero che compongono la media collettiva

Cliccando l'icona della lente di ingrandimento compare una barra di ricerca che permette di cercare una specifica valutazione all'interno del database o del telefono inserendo il numero telefonico oppure il nome del contatto se presente in rubrica. Attraverso questa funzione è inoltre possibile inserire una valutazione per uno specifico numero di telefono semplicemente indicandolo nella barra di ricerca.

Per inserire o modificare una valutazione è sufficiente cliccare sulla riga corrispondente al numero di telefono.

Cliccando sull'icona del profilo in alto a sinistra si può invece accedere alla schermata di impostazioni del profilo utente per modificare le preferenze.

2.4. Schermata delle Valutazioni

Attraverso questa schermata è possibile inserire o modificare una valutazione di un cliente. Ogni valutazione si compone di un rating da 0 a 5, un commento personale e una valutazione collettiva, con eventuali commenti di altre persone.

L'utente ha la possibilità di scrivere un commento e di decidere se renderlo pubblico, quindi visibile anche agli altri, oppure tenerlo per sé attraverso la spunta "Rendi pubblico il commento".

Al di sotto della valutazione collettiva vengono mostrati gli eventuali commenti fatti da altre persone riguardo quel cliente.

Cliccando su conferma la valutazione viene caricata sul database e l'utente viene riportato alla home page aggiornata con il nuovo rating.

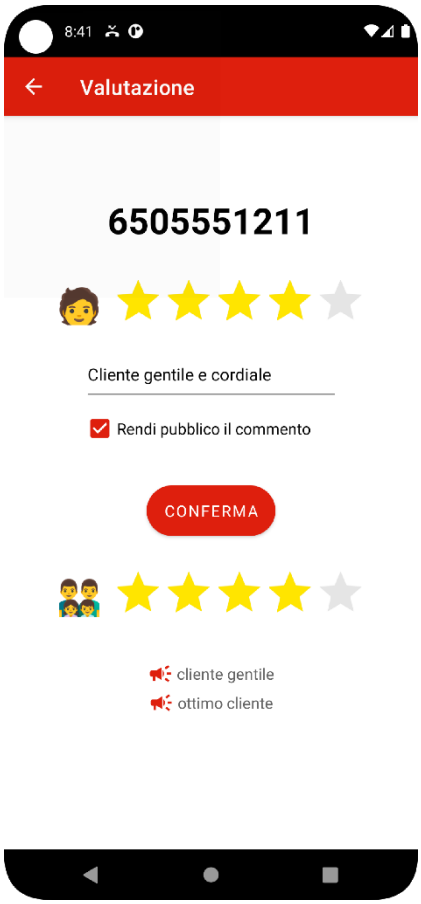

Figura 6: Valutazione

2.5. Impostazioni Profilo

Nella schermata corrente vengono mostrate le impostazioni principali per l'utente.

Al di sotto del logo BestClient vengono mostrati e-mail e partita IVA con cui l'utente si è registrato, mentre nella seconda parte della schermata vengono messe a disposizione alcune funzionalità per l'utente, come la scelta della modalità di notifica (Notifica classica, Toast Message o Popup), il pulsante di logout e il pulsante per eliminare l'utente dall'applicazione. Cliccando su quest'ultimo verrà chiesta un'ulteriore conferma all'utente e dopodiché l'utente verrà eliminato dal database con anche le sue valutazioni.

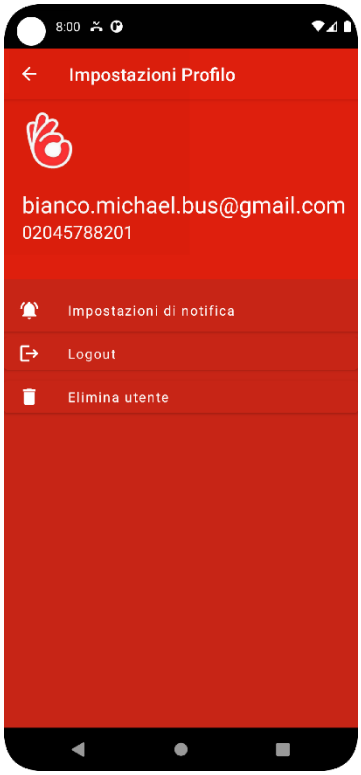

Figura 7: Impostazioni

2.6. Notifica

Una volta che l'utente riceve una chiamata vengono recuperati i dati della valutazione personale e collettiva relativi al numero chiamante e vengono mostrati a seconda della modalità di notifica scelta. In figura è mostrato il caso di una notifica classica.

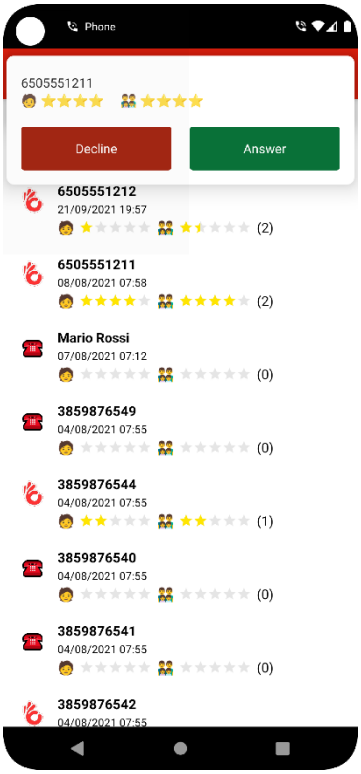

Figura 8: Notifica

3. Strumenti utilizzati

L'applicazione è stata sviluppata in Java attraverso l'editor Android Studio, principale piattaforma di sviluppo per applicazioni Android, mentre per il database è stato utilizzato Firebase, piattaforma messa a disposizione da Google. Per la gestione e il salvataggio delle varie versioni del software è stato utilizzato GitHub, mentre WayScript permette l'aggiornamento automatico del database.

Figura 9: Strumenti utilizzati

3.1. Android Studio

Attraverso Android Studio è stato possibile scrivere il codice in linguaggio Java, costruire l'interfaccia grafica delle varie schermate e testare l'applicazione attraverso un apposito simulatore. Alla fine del testing è stato possibile creare una prima versione beta da caricare sullo store di Google in modo da metterla a disposizione ai vari utenti per avere un feedback sugli aspetti da correggere o da migliorare.

3.2. Firebase

Il database Firebase di Google ha una struttura gerarchica ad albero, a differenza dei tradizionali database relazionali, e offre, attraverso un'apposita API, funzioni per leggere e scrivere sul database in realtime. Offre inoltre un meccanismo di autenticazione per limitare l'accesso degli utenti al database e a parti di esso, in particolare nel caso di BestClient gli utenti possono solo leggere e scrivere il ramo di database riservato a loro e leggere la versione sintetica del db, mentre gli amministratori hanno completo accesso.

Il database di BestClient è organizzato in tre nodi principali dopo quello di root: il nodo "Users"; il nodo "lastUpdate" per memorizzare il timestamp dell'ultimo aggiornamento automatico del database; il nodo "ratingAVG".

3.2.1. Users

Il ramo "Users" contiene tanti rami quanti sono gli utenti registrati, ognuno dei quali è identificato da un codice univoco all'interno del database e contiene tutte le informazioni sull'utente. In particolare memorizza la sua e-mail, la partita IVA e un ramo "Valutazioni", che contiene per ogni numero la valutazione data, l'eventuale commento, un flag che indica se il commento è pubblico oppure privato e il timestamp di quando è avvenuto il caricamento della valutazione.

Figura 10: Esempio struttura database, Users

3.2.2. RatingAVG

Il ramo “RatingAVG” contiene una versione sintetica del database, raccoglie per ogni numero valutato una lista di commenti pubblici rilasciati dai vari utenti, la valutazione media complessiva e il numero di utenti che hanno contribuito a calcolare quella media. È molto utile perché consente in un attimo di risalire alla valutazione media di un cliente, senza dover attraversare ogni ramo dei vari utenti per cercare il numero corrispondente.

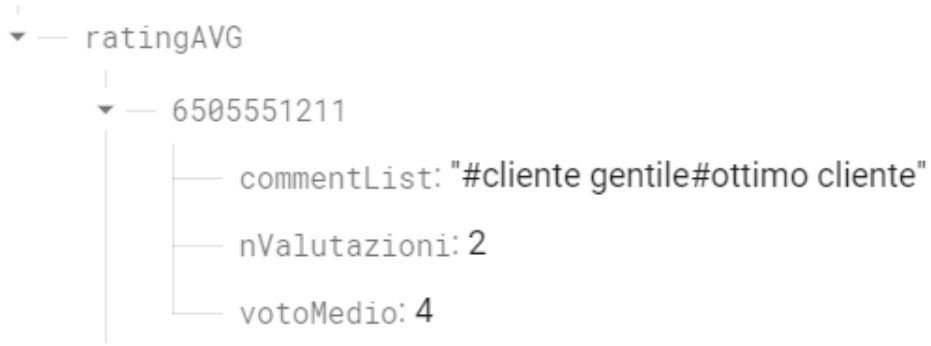

Figura 11: Esempio struttura database, ratingAVG

3.3. GitHub

Attraverso il repository gratuito di GitHub abbiamo potuto salvare le varie versioni del software nel Cloud in modo da renderle disponibili a entrambi gli sviluppatori man mano che l’applicazione veniva creata. Il repository si trova a questo link https://github.com/Tod-dev/Best-Client-v2 ed è un repository privato, modificabile solo da noi sviluppatori.

3.4. WayScript

La piattaforma WayScript permette di eseguire uno script python che consente in automatico, una volta al giorno, di scaricare tutti i dati del ramo Users per raggrupparli in una versione sintetica da mostrare nel ramo ratingAVG. In particolare, vengono estratti per ogni utente le sue valutazioni e gli eventuali commenti pubblici assegnati ad ogni numero telefonico, dopodichè viene fatta una media del voto, vengono raccolti gli ultimi cinque commenti pubblici e vengono scritti nel ramo ratingAVG separati dal simbolo #.

3.4.1. Aggiornamento del database

Lo script python per aggiornare il database è costituito da due funzioni principali:

- La funzione *fetch_data()*, la quale si collega al ramo Users del database e, per ogni utente, cerca le valutazioni assegnate, memorizzando in una lista tutte le informazioni relative, come il rating assegnato, il numero di telefono, la data di assegnazione, il commento e il flag che indica la visibilità del commento. Dopo aver raccolto tutti i dati scorre i vari numeri telefonici e per ognuno salva la lista degli ultimi cinque commenti pubblici, insieme alla media dei voti e al numero di valutazioni. Una volta costruita la struttura sintetica viene ritornato un dizionario python con chiave il numero di telefono e come elemento la struttura sintetica delle sue valutazioni.

```python
def fetch_data():
    endpoint = db.reference("/Users")
    users = endpoint.get()
    KEY = 'Valutazioni'
    ratingList = []
    for k in users:
        if KEY in users[k]:
            for num in users[k][KEY]:
                pubblica = False
                if 'pubblica' in users[k][KEY][num]:
                    pubblica = users[k][KEY][num]['pubblica']
                r = RatingBig(users[k][KEY][num]['commento'], num, users[k][KEY][num]['date'],
                              users[k][KEY][num]['voto'], pubblica)
                ratingList.append(r.__dict__)

    df = pd.DataFrame(ratingList)

    numbers_data = {}
    grouped_df = df.groupby('numero')
    for key, item in grouped_df:
        group = grouped_df.get_group(key).sort_values(by='date', ascending=False)

        commentlist = []
        cont_comments = 0
        max_comments = 5
        for i, row in group.iterrows():
            if (cont_comments == max_comments):
                break
            if row['commento'] not in ['', 'commento'] and row['pubblica']:
                cont_comments += 1
                comment = (row['commento'][:50] + '..') if len(row['commento']) > 30 else row['commento']
                commentlist.append(comment)

        comments_str = ' '.join(['#{}'.format(c) for c in commentlist])

        count = 0
        for i in group['voto']:
            count = count + 1

        numbers_data[key] = {
            'grade': group['voto'].mean(),
            'comments': comments_str,
            'nValutazioni': count,
        }
    return numbers_data
```

Figura 12: Funzione fetch_data()

- La funzione *update_data()*, responsabile del caricamento della struttura sintetica sul database. Questo metodo prende il riferimento al ramo ratingAVG, crea tanti rami quanti sono i numeri di telefono contenuti nella struttura recuperata dalla funzione di fetch precedente e in ognuno inserisce i campi valutazione media, lista di commenti e numero di valutazioni.

```
def update_data(numbers_data):
    ref = db.reference("/ratingAVG")
    for number in numbers_data:
        r = RatingAvg(numbers_data[number]['grade'], numbers_data[number]['comments'],
                      numbers_data[number]['nValutazioni'])
        ref.child(number).set(r.__dict__)

    ref = db.reference("/lastUpdate")
    dt_string = datetime.datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    ref.set(dt_string)
    return
```

Figura 13: Funzione update_data()

4. Struttura del codice

4.1. Use case diagram

Lo Use case diagram raccoglie le principali interazioni che l'utente può avere con l'applicazione, tra le principali possiamo vedere l'inserimento o la modifica di una valutazione, la visualizzazione dell'elenco dei numeri e delle valutazioni, oppure la possibilità di scegliere la modalità di notifica, insieme alla sua ricezione quando il telefono squilla.

Si può notare che un utente non registrato o che non ha fatto il login non può in alcun modo utilizzare l'applicazione.

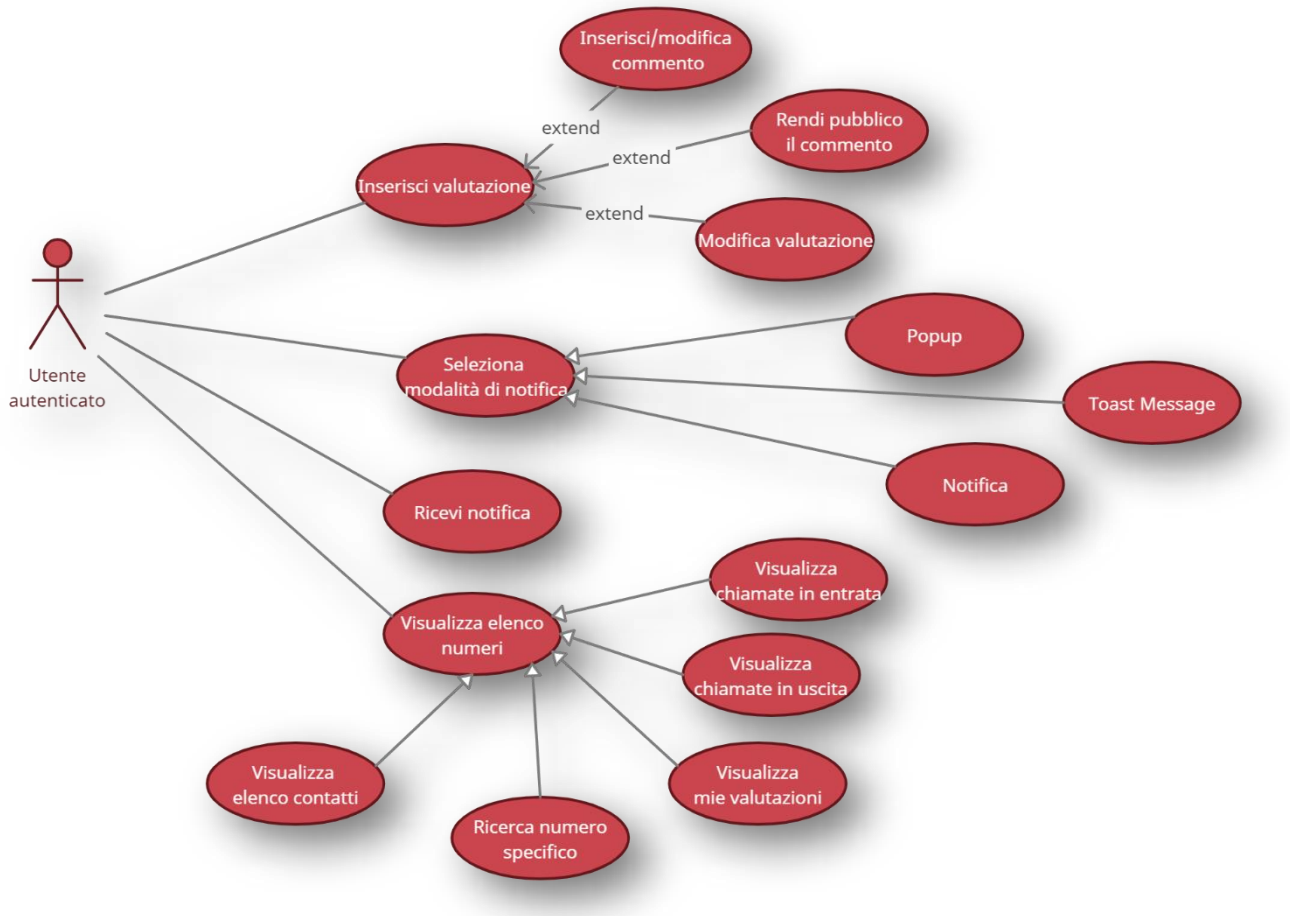

Figura 14: Use case diagram

4.1.1. Inserimento/modifica di una valutazione

L'utente può inserire una nuova valutazione attraverso l'apposita schermata selezionando un numero dall'elenco visualizzato nella home page, oppure può modificare la stessa cambiando il voto, inserendo o modificando un commento, oppure cambiando la spunta pubblico/privato relativa al commento.

Nome	Inserimento di una valutazione
Attore partecipante	Utente autenticato
Precondizioni	L'utente deve essersi autenticato
Flusso degli eventi	1. Selezione del numero da valutare tramite la home page 2. Inserimento/modifica di voto, commento o spunta relativa alla visibilità del commento 3. Invio dei dati al database dopo il clic sul bottone di conferma
Casi particolari	4. Modifica della valutazione 5. Inserimento/modifica del commento 6. Modifica spunta commento pubblico o privato

4.1.2. Selezione della modalità di notifica

L'utente ha la possibilità di scegliere la notifica più adatta alle sue esigenze accedendo alla schermata del profilo e selezionando una tra le modalità disponibili, come notifica classica, popup o toast message. La scelta di default è la notifica classica.

Nome	Selezione modalità di notifica
Attore partecipante	Utente autenticato
Precondizioni	L'utente deve essersi autenticato
Flusso degli eventi	1. Selezione della modalità di notifica attraverso la schermata di impostazioni profilo 2. Ricezione di una chiamata e visualizzazione della notifica secondo la modalità scelta
Specializzazioni	3. Notifica classica 4. Popup 5. Toast Message

4.1.3. Visualizzazione dell’elenco numeri

Attraverso la home page vengono visualizzati i numeri letti dal registro chiamate in entrata/uscita, dalla rubrica o dal database, a seconda di quale filtro ha impostato l’utente. C’è la possibilità di ricercare anche un numero specifico inserendolo nella barra di ricerca sia in formato numerico sia come nome/cognome se questo è associato ad un contatto salvato in rubrica, oppure l’utente può inserire una valutazione ad un numero che non visualizza nell’elenco nel caso in cui abbia eliminato la voce nel registro chiamate o nel caso in cui il cliente abbia chiamato su un telefono diverso da quello su cui è installata l’applicazione.

Nome	Visualizzazione elenco numeri
Attore partecipante	Utente autenticato
Precondizioni	L’utente deve essersi autenticato
Flusso degli eventi	1. Impostazione del filtro nella home page
Specializzazioni	2. Visualizzazione elenco chiamate in entrata 3. Visualizzazione elenco chiamate in uscita 4. Visualizzazione elenco contatti in rubrica 5. Visualizzazione “le mie valutazioni” presenti sul database 6. Ricerca di un numero specifico

4.2. Class Diagram

Vediamo ora le classi che compongono il software e le loro relazioni attraverso un class diagram.

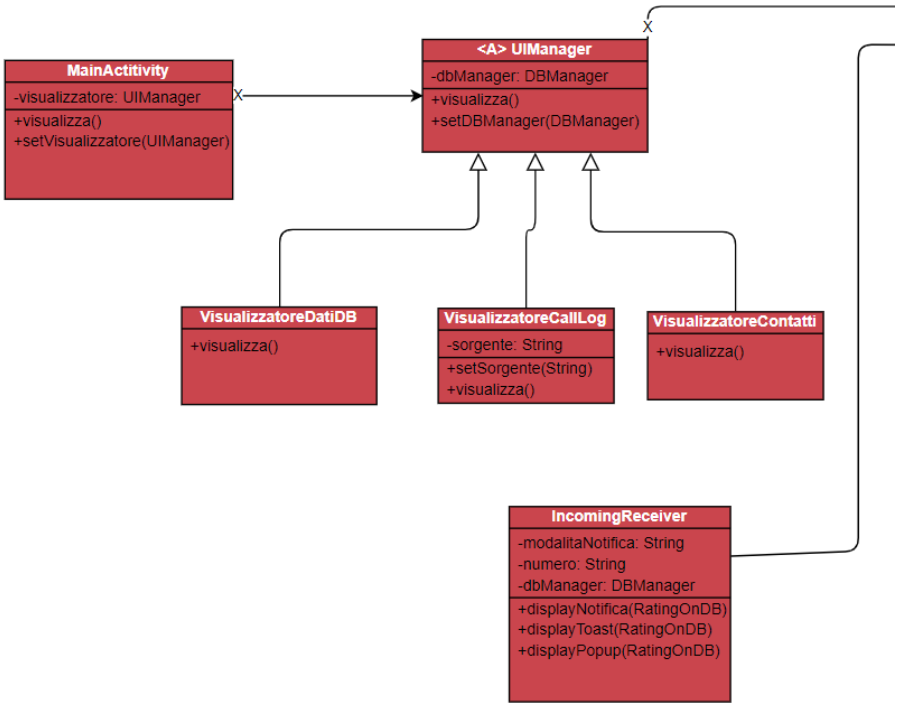

Figura 15: Class Diagram, Strategy

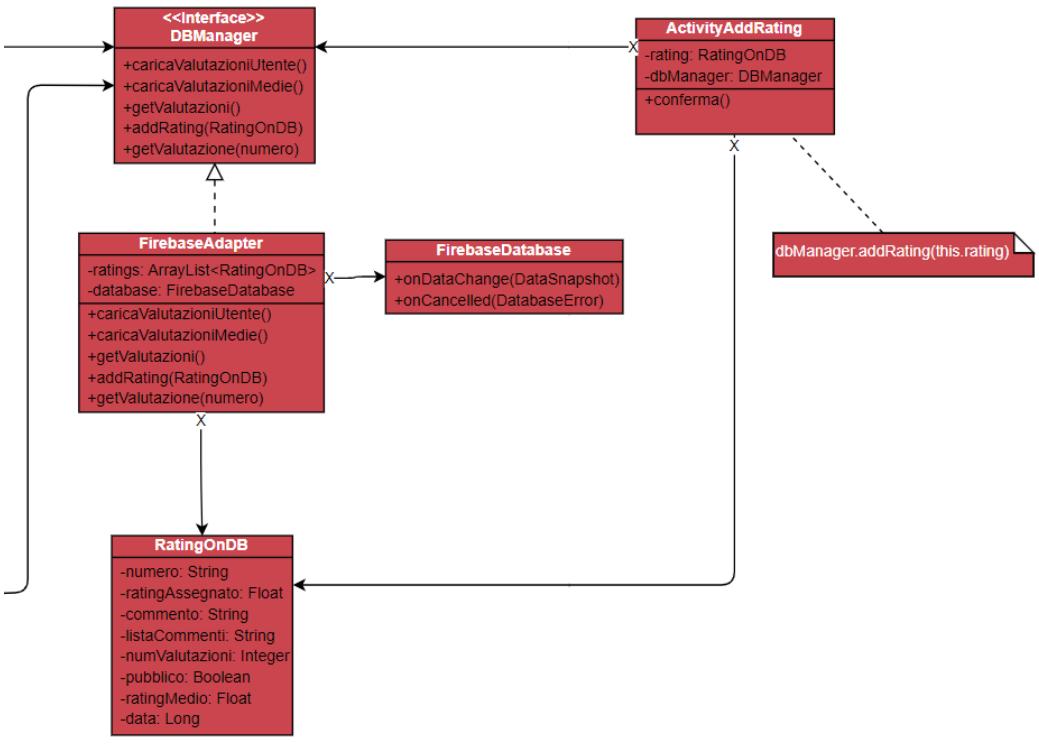

Figura 16: Class Diagram, Adapter

Il diagramma è stato diviso in due parti per facilitarne la lettura ed evidenziare l'uso di due design pattern principali, uno Strategy e un Adapter, che verranno descritti nel prossimo paragrafo. Si può notare che il principio di Dependency Inversion è stato rispettato in quanto le classi dipendono per lo più da interfacce.

Nel class diagram si notano alcune classi principali:

- **MainActivity** è responsabile della visualizzazione nella home page dell'elenco di valutazioni che l'utente ha dato o deve ancora dare, in particolare mostra i rating relativi al registro chiamate in entrata/uscita, i rating dati dall'utente e quelli relativi ai contatti della rubrica a seconda del filtro scelto.
- **IncomingReceiver** è la classe che si attiva nel momento in cui arriva una chiamata in entrata, legge il numero telefonico che sta chiamando e chiede al gestore del database di recuperare i dati ad esso relativi, dopodiché se il numero è presente all'interno del database vengono mostrate le valutazioni personali e collettive attraverso una notifica scelta dall'utente, mentre non viene mostrato nulla se il numero non è nel database.
- **DBManager** è il gestore del database e contiene tutte le funzioni utili per recuperare informazioni sui numeri valutati, come il download di tutte le valutazioni fatte dall'utente o la ricerca di un rating dato un numero di telefono specifico.
- **RatingOnDB** rappresenta ogni rating e contiene tutte le informazioni ad esso relative, tra cui: il numero di telefono; la valutazione; un eventuale commento e il relativo flag per sapere se è pubblico o privato; la valutazione media collettiva e il numero di utenti che hanno contribuito al calcolo; la lista dei commenti pubblici dati da altre persone; la data in cui è stato valutato quel numero.
- **ActivityAddRating** è la classe che si occupa dell'assegnazione di una valutazione, prende tutti i dati compilati dall'utente nell'apposita schermata e, una volta cliccato il pulsante di conferma, invoca una funzione del gestore del database per caricare o aggiornare la nuova valutazione.

4.3. Sequence Diagram

Analizziamo ora come si comporta il software e come interagiscono tra di loro le classi in due situazioni tipiche come l'inserimento di una valutazione e la chiamata in arrivo.

4.3.1. Inserimento di una valutazione

Una volta che l'applicazione è in uso viene fatto partire un thread che carica i dati delle valutazioni dal database remoto e nel frattempo viene visualizzato l'ultimo elenco numeri selezionato (nell'esempio viene mostrato il caso del registro chiamate in entrata).

Quando è finito il caricamento dei dati viene aggiornata la user interface con le valutazioni personali e collettive dei numeri mostrati.

Una volta selezionato il numero da valutare si apre la schermata di inserimento/modifica valutazione e, una volta compilata, questa viene caricata sul database e si ritorna all'elenco dei numeri precedente con la valutazione aggiornata.

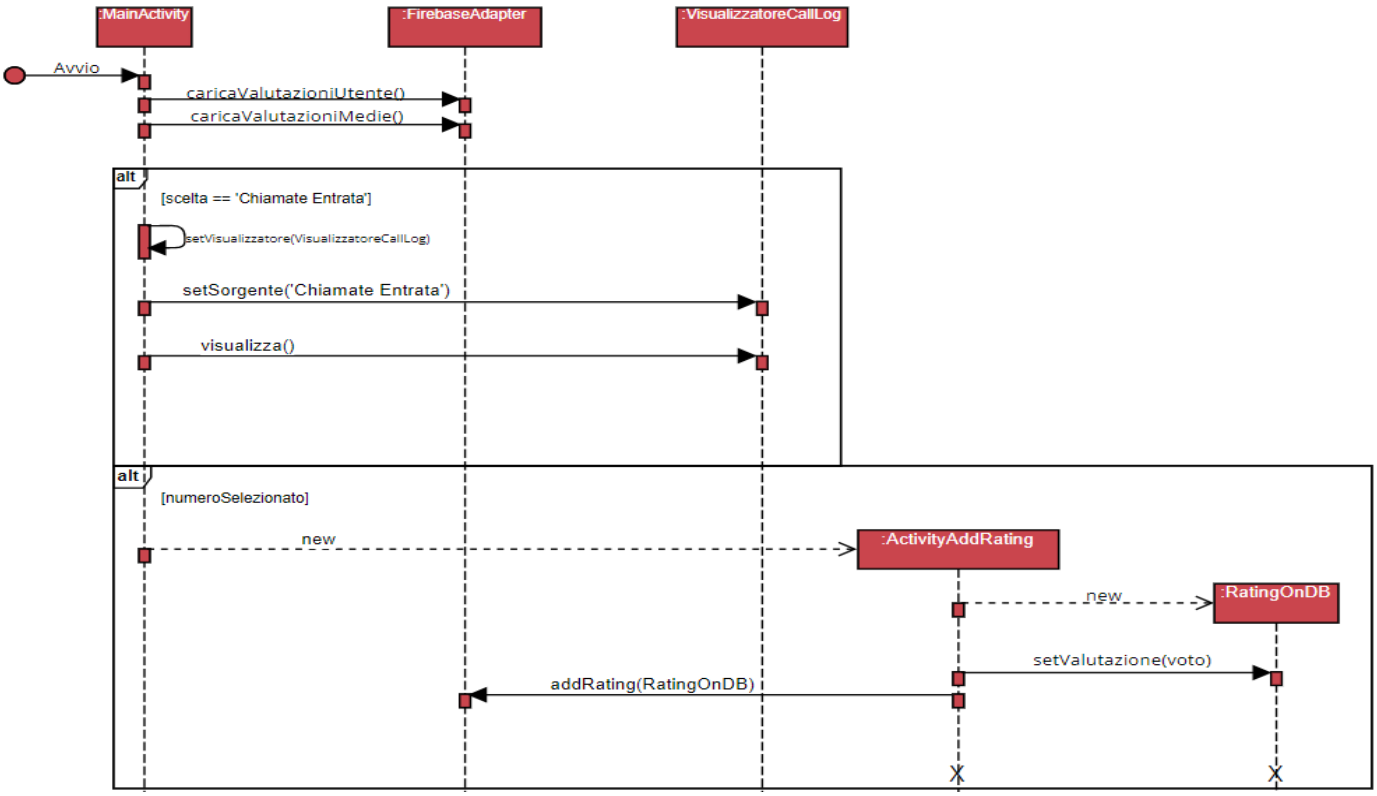

Figura 17: Sequence Diagram - Inserimento Valutazione

4.3.2. Chiamata in arrivo

All'arrivo di una chiamata la classe IncomingReceiver risale al numero chiamante e chiede al gestore del database di inviargli la valutazione corrispondente. Se il numero è presente nel database, ovvero è presente almeno una tra le valutazioni dell'utente e le valutazioni collettive, viene recuperata la preferenza della modalità di notifica e, in base a quella, viene mostrato il messaggio corrispondente contenente le valutazioni disponibili per quel numero. Mentre se il numero non è mai stato valutato né dall'utente corrente né da nessun altro utente non viene mostrata alcuna notifica.

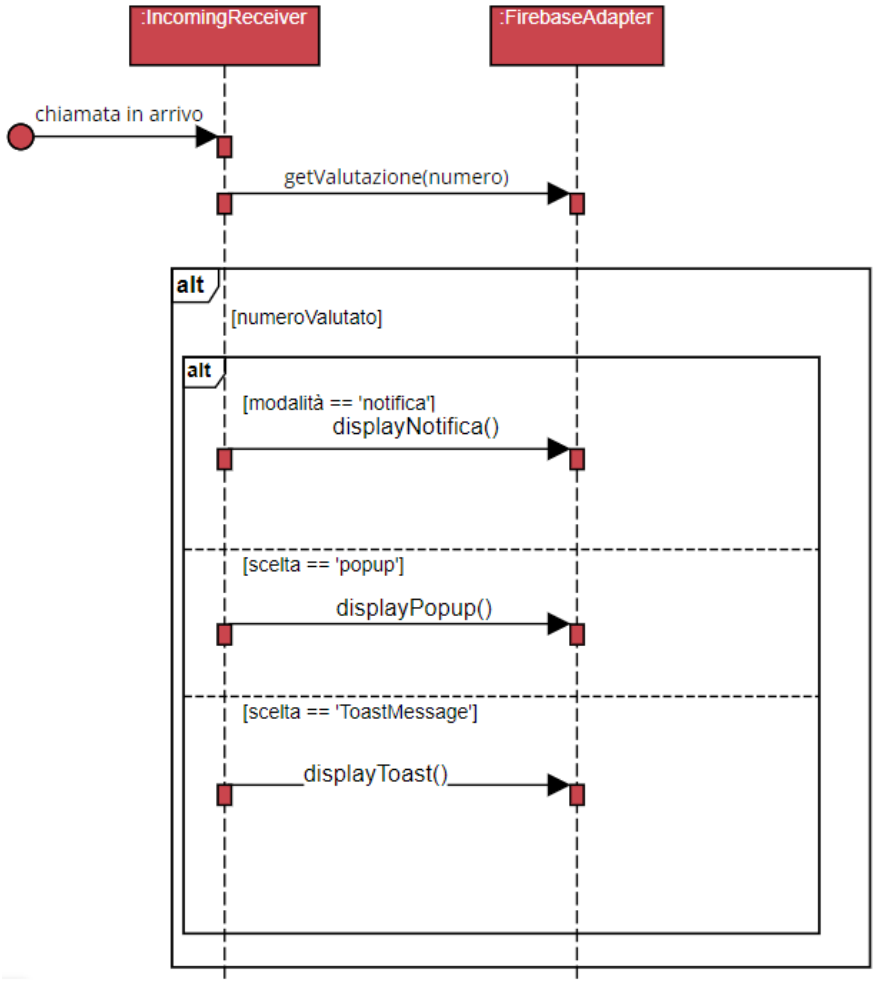

Figura 18: Sequence Diagram - Chiamata in arrivo

4.4. Activity Diagram

Di seguito vengono rappresentati gli activity diagram delle due attività principali, quali l'inserimento di una valutazione e la ricezione di una notifica quando arriva una chiamata. Si può notare che se il numero non è presente sul database, ovvero non è ancora stato valutato, la notifica non viene mostrata.

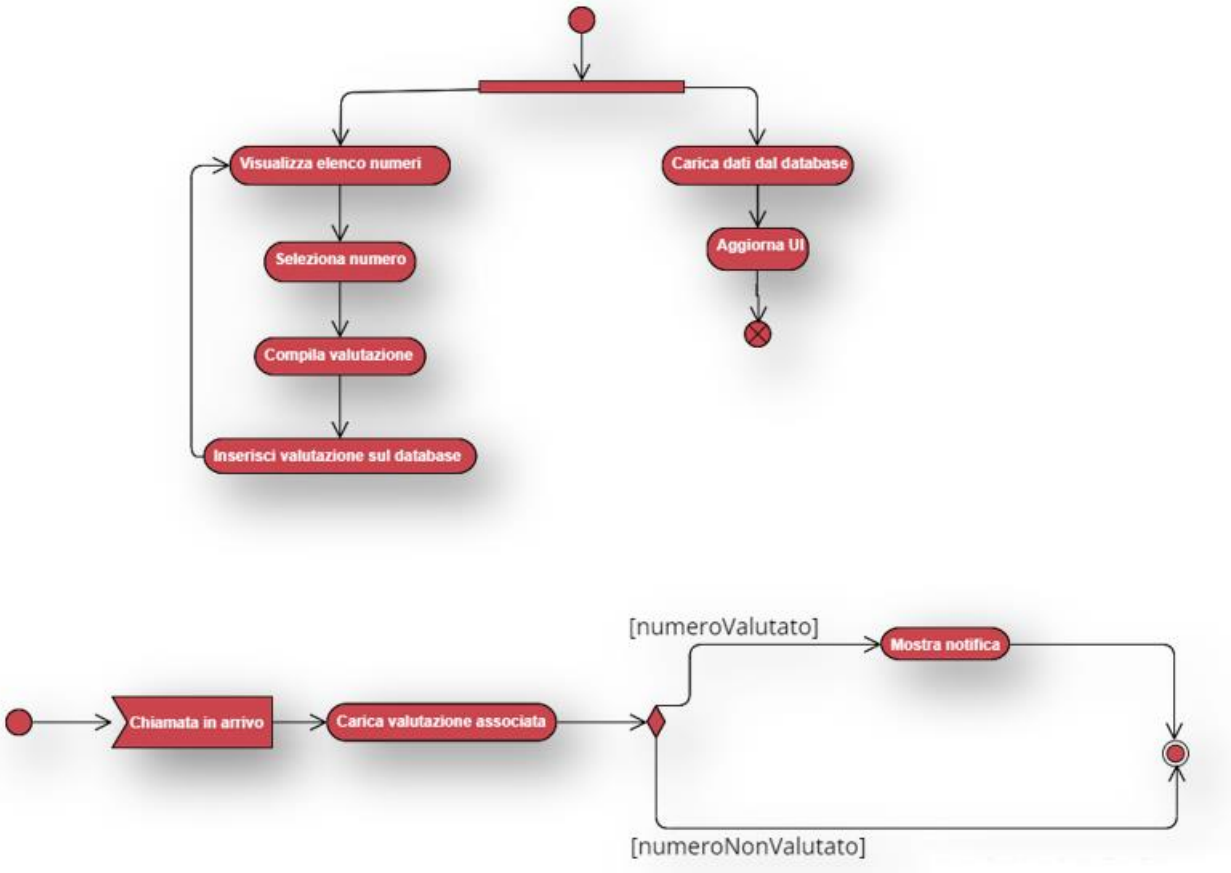

Figura 19: Activity Diagram

4.5. Implementazione

La parte funzionale dell'applicazione è scritta interamente in Java, mentre la parte grafica è scritta attraverso il linguaggio a marcatori xml. Ogni schermata è rappresentata da una Activity, una classe specifica di Android Studio che crea una pagina in cui viene visualizzato il contenuto xml grafico del file ad essa associato e che permette l'interazione con l'utente. A fianco delle Activity principali ci sono le classi viste nel paragrafo 4.2 relativo al Class Diagram che offrono funzionalità ad hoc per ogni esigenza da realizzare nell'applicazione, dalla rappresentazione delle valutazioni mediante la classe RatingOnDB alla gestione del database mediante la classe DBManager. Per accedere a determinate funzioni di Android come la lettura del registro chiamate, della rubrica o la rilevazione di una chiamata sono necessarie alcuni metodi e librerie descritte nei prossimi paragrafi.

4.6. Rilevazione di una chiamata

In Android è possibile intercettare una chiamata in arrivo creando una classe che estenda BroadcastReceiver, componente che permette di attivare il relativo metodo *onReceive()* per gestire le operazioni disponibili su un determinato evento a cui ci siamo registrati una volta che questo si è verificato.

Ogni operazione è denominata Intent e, nel caso specifico di BestClient, si attiva la funzionalità di notifica solo quando l'evento verificatosi corrisponde ad una chiamata in arrivo. Una volta intercettata la chiamata viene istanziato un oggetto di classe CallListener, un componente privato che estende la classe PhoneStateListener per accedere ai dati relativi alla chiamata, tra i quali il numero di telefono di chi sta chiamando. Attraverso il numero viene richiesta la valutazione corrispondente al gestore del database e, se il numero è presente in quanto esiste almeno una valutazione, viene letta la preferenza sulle modalità di notifica scelta dall'utente e viene mostrato il messaggio relativo, mentre se il numero non ha valutazioni non viene mostrata alcuna notifica.

```
public void onReceive(Context context, Intent intent) {
    if(ContextCompat.checkSelfPermission(context, Manifest.permission.READ_PHONE_STATE) == PackageManager.PERMISSION_GRANTED){
        if (intent.getStringExtra(TelephonyManager.EXTRA_STATE).equals(TelephonyManager.EXTRA_STATE_RINGING)) {
            /*THE PHONE IS RINGING*/
            this.context = context;
            TelephonyManager tm = (TelephonyManager) context.getSystemService(Context.TELEPHONY_SERVICE);
            CallListener cl = new CallListener();
            tm.listen(cl, PhoneStateListener.LISTEN_CALL_STATE);

        } else if (intent.getStringExtra(TelephonyManager.EXTRA_STATE).equals(TelephonyManager.EXTRA_STATE_IDLE)) {
            Log.d("CALL: ", "CALL ENDED");
        }
    }
    else Toast.makeText(context,"This app couldn't read phone state, please allow in settings", Toast.LENGTH_LONG).show();

}

private class CallListener extends PhoneStateListener{

    @RequiresApi(api = Build.VERSION_CODES.O)
    public void onCallStateChanged(int state, String incomingNumber){

        if (incomingNumber.length() > 0 && state == TelephonyManager.CALL_STATE_RINGING) {
            sp = context.getSharedPreferences("UserPreferences", Context.MODE_PRIVATE);
            String email = sp.getString("email", "");
            String password = sp.getString("password", "");

            if(email.equals("") || password.equals("")){
                return;
            }

            /* ESTRACT THE NUMBER OF THE CALLER */
            String onlyNumber=incomingNumber;
            /*IF THE number w the PREFIX >= 10 -> we read only the LAST 10 character */
            if(incomingNumber.length() >= 10)
                onlyNumber = incomingNumber.substring(incomingNumber.length() - 10);

            getRatingFromNumber(onlyNumber);
        }
    }
```

Figura 20: IncomingReceiver

Dal codice si possono notare tre aspetti principali:

- Per poter leggere lo stato delle chiamate è necessario che l'utente abbia acconsentito al permesso di leggere lo stato del telefono, altrimenti non è possibile intercettare le chiamate;
- La notifica viene mostrata solo se l'utente ha eseguito il login, ovvero sono noti all'applicazione il suo username e la sua password;
- Vengono estratti gli ultimi dieci caratteri del numero in modo tale da eliminare il prefisso se presente ed evitare così problemi di lettura del numero da parte del gestore del database.

La funzione *getRatingFromNumber()* chiede semplicemente al database manager la valutazione relativa al numero chiamante e mostra la notifica opportuna.

4.7. Accesso ai contatti

La lettura dei contatti presenti nel telefono viene fatta attraverso una funzione dedicata *fetchContacts()*, invocata in background all'avvio dell'applicazione in modo da poter mostrare nelle varie schermate i nomi delle persone valutate al posto del relativo numero di telefono, e può avvenire solo se l'utente ha acconsentito al permesso di leggere la rubrica. Ogni contatto viene salvato in un'HashMap con chiave il numero telefonico e con valore il nome del contatto, in modo da velocizzare la lettura, soprattutto in presenza di rubriche molto lunghe, e viene utilizzato un set di appoggio per evitare il caso di contatti duplicati. La funzione restituisce i contatti dentro un'ArrayList per poterli caricare più rapidamente dentro alla ListView che li mostra a video.

```
public static List<Contact> fetchContacts(ContentResolver contentResolver, Context k){
    Set<Contact> contacts = new TreeSet<>();    //con un set impedisco l'inserimento di contatti duplicati
    Map<String,String> contactMap = new HashMap<>();

    if ((ContextCompat.checkSelfPermission(k, Manifest.permission.READ_CONTACTS) == PackageManager.PERMISSION_GRANTED)) {
        //solo se ho i permessi di accesso alla rubrica
        try (Cursor cursor = contentResolver.query(
                ContactsContract.Contacts.CONTENT_URI
                , null
                , null
                , null
                , null)) {
            if (cursor != null && cursor.getCount() > 0) {
                while (cursor.moveToNext()) {
                    Contact contact = new Contact();
                    String contact_id = cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts._ID));
                    contact.setName(cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)));
                    int hasPhoneNumber = Integer.parseInt(cursor.getString(cursor.getColumnIndex(
                                                          ContactsContract.Contacts.HAS_PHONE_NUMBER)));
                    if (hasPhoneNumber > 0) {
                        Cursor phoneCursor = contentResolver.query(
                                ContactsContract.CommonDataKinds.Phone.CONTENT_URI
                                , null
                                , ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = ?"
                                , new String[]{contact_id}
                                , null);
                        if (phoneCursor != null) {
                            phoneCursor.moveToNext();
                            String phoneinContact = phoneCursor.getString(phoneCursor.getColumnIndex(
                                                                  ContactsContract.CommonDataKinds.Phone.NUMBER));
                            if(!contactMap.containsKey(phoneinContact)) {
                                //non è stato trovato un contatto con stesso numero
                                contact.setPhone(filterOnlyDigits(phoneinContact));
                                contacts.add(contact);
                                contactMap.put(contact.getPhone(), contact.getName());
                            }
                        }
                        if (phoneCursor != null) phoneCursor.close();
                    }
                }
            }
        } catch (Exception ignored) {}
    }
    return new ArrayList<>(contacts);
}
```

Figura 21:FetchContacts()

4.8. Accesso al registro chiamate

L'accesso al registro chiamate avviene all'interno della classe visualizzatoreCallLog ed è possibile solo se l'utente ha dato il consenso alla lettura dello stesso. La funzione dedicata si chiama *getCallLog()* ed è in grado di distinguere tra chiamate in entrata e chiamate in uscita a seconda della pagina selezionata. Il metodo legge attraverso un cursore tutte le chiamate, ordinate dalla più recente alla più vecchia, filtrandole tra ricevute ed effettuate, dopodiché per ogni riga letta verifica di non aver già trovato quel numero, in modo da evitare di riscrivere più volte la stessa riga a video, e infine controlla se i numeri trovati corrispondono ad un contatto in rubrica e se possiedono una valutazione personale e collettiva per mostrare le informazioni a video.

```
public static List<Rating> getCallLog(Context context) {
    int scelta = Integer.parseInt(sp.getString("scelta", String.valueOf(CHIAMATE_ENTRATA)));
    Cursor c = context.getContentResolver().query(CallLog.Calls.CONTENT_URI,
                                                  new String[]{"number", "date", "type"}, null, null, "date DESC");
    int colNumber = c.getColumnIndex(CallLog.Calls.NUMBER);
    int colDate = c.getColumnIndex(CallLog.Calls.DATE);
    List<Rating> ratingsRet = new ArrayList<>();
    List<RatingCallLog> ratingCallLogs = new ArrayList<>();
    while (c.moveToNext()) {
        String type = c.getString(c.getColumnIndex(CallLog.Calls.TYPE));
        int typeNumber = Integer.parseInt(type);
        String number = c.getString(colNumber);
        Date date = new Date(Long.parseLong(c.getString(colDate)));
        RatingCallLog check = new RatingCallLog(number, date);
        callLogNumbers.add(check);

        if ((scelta == CHIAMATE_ENTRATA && typeNumber == CallLog.Calls.OUTGOING_TYPE) ||
                (scelta == CHIAMATE_USCITA && typeNumber != CallLog.Calls.OUTGOING_TYPE))
            continue;

        boolean checkIfExist = false;
        for (RatingCallLog r : ratingCallLogs) {
            boolean res = r.group_by(check);
            if (res) {
                checkIfExist = true;
                break;
            }
        }
        if (!checkIfExist) {
            ratingCallLogs.add(check);
            Rating r;

            if (contactMap.containsKey(check.getNumero()))
                r = new Rating(check.getNumero(), contactMap.get(check.getNumero()));
            else r = new Rating(check.getNumero());

            Date last = new Date(check.getDate());
            r.setDate(last.getTime());

            if (ratingsOnDb.containsKey(r.getNumero()))
                r.setVoto(ratingsOnDb.get(r.getNumero()).getVoto());

            ratingsRet.add(r);
        }
    }
    c.close();
    return ratingsRet;
}
```

Figura 22: Lettura registro chiamate

4.9. Accesso al database

Firebase è un database realtime in grado di aggiornare automaticamente i dati letti una volta che questi sono cambiati, senza doverli rileggere. Per collegarsi ad un ramo specifico è sufficiente istanziare un oggetto di classe FirebaseDatabase, attraverso il quale si può ottenere il collegamento alla radice del database e, tramite esso, si crea un oggetto di classe DatabaseReference mediante il quale si può selezionare il ramo che si vuole leggere o scrivere.

Per leggere dal database bisogna aggiungere un listener sull'oggetto appena creato e sovrascrivere la funzione *onDataChange()*, attraverso cui viene ritornato un Datashot, una struttura annidata contenente tutti i dati da quel ramo fino alle foglie. Se il nodo che vogliamo leggere è una foglia, per ottenere i suoi dati si può invocare il metodo *getValue()* del Datashot, a cui viene passata la classe che rappresenta l'oggetto e che deve avere ogni attributo con stesso nome delle chiavi del nodo.

Per scrivere invece bisogna ottenere ancora il riferimento di tipo DatabaseReference al punto in cui vogliamo effettuare una modifica e successivamente invocare il suo metodo *setValue()* passandogli l'oggetto da scrivere. Il database scriverà ogni attributo dell'oggetto come chiave:valore, dove la chiave è il nome dato all'attributo e il valore è il suo contenuto.

```
String uid = context.getSharedPreferences("UserPreferences", Context.MODE_PRIVATE).getString("uid", "");

FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference ratingsRef = database.getReference("Users").child(uid).child("Valutazioni");

ratingsRef.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

        for (DataSnapshot d : dataSnapshot.getChildren()) {
            RatingOnDB r = d.getValue(RatingOnDB.class);
            r.setNumero(d.getKey());
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        // Failed to read value
        Log.w("Main", "Failed to read value.", error.toException());
    }
});
```

Figura 23: Esempio lettura dal database

```
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference(USERS).child(uid).child(VALUTAZIONI).child(phoneNumber);

myRef.setValue(ratingOnDb);
```

Figura 24: Esempio scrittura sul database

4.10. SharedPreferences

Le SharedPreferences sono componenti di Android Studio che permettono di salvare stringhe in formato chiave:valore accessibili da ogni schermata dell'applicazione, senza doverle passare tra una schermata e l'altra. Sono utili per memorizzare i dati di login come username, password, partita IVA e le preferenze dell'utente, come la scelta della schermata da visualizzare tra le chiamate in entrata, in uscita, le valutazioni dell'utente e la rubrica telefonica, oppure la modalità di notifica scelta tra notifica classica, popup o toast message.

Si accede ad esse tramite un oggetto di classe SharedPreferences, da cui si può ottenere anche un Editor apposito per modificarle. La lettura avviene tramite il metodo *getString()* di SharedPreferences, al quale bisogna passare la chiave dell'elemento da leggere e un valore di default da assegnare nel caso in cui l'elemento non esista. Per la scrittura bisogna invocare il metodo *putString()* dell'editor passando come parametri la chiave e il valore da scrivere.

```
SharedPreferences sp = getSharedPreferences("UserPreferences", Context.MODE_PRIVATE);

//SCRITTURA
SharedPreferences.Editor editor = sp.edit();

editor.putString("email", "");
editor.putString("password", "");
editor.putString("piva", "");
editor.putString("uid", "");

editor.putString("notificationPreference", String.valueOf(R.id.notification));
editor.putString("scelta", String.valueOf(HomeActivity.CHIAMATE_ENTRATA));
editor.apply();

//LETTURA
String email = sp.getString("email", "");
String password = sp.getString("password", "");
```

Figura 25: Esempio scrittura e lettura SharedPreferences

5. Refactoring

5.1. Design Pattern

I design pattern sono delle “best practice”, degli schemi utilizzati per la costruzione di software di qualità, maneggevole e facile da modificare in caso di cambiamenti. Si basano sui princìpi della programmazione ad oggetti quali incapsulamento, ereditarietà e polimorfismo e riescono a descrivere un problema e le sue soluzioni in un modo che può essere utilizzato molte volte, oltre a dare un’idea immediata a qualcuno di esterno che legge il software di come sono strutturate e collegate le classi. Attraverso il nome del design pattern è possibile, infatti, descrivere un problema di design e il suo contesto, rappresentare le sue soluzioni attraverso gli elementi principali che costituiscono il pattern insieme alle loro relazioni, e infine indicare le conseguenze, ovvero i risultati che si possono ottenere applicando quello specifico design pattern.
I pattern possono essere classificati secondo due tipologie e tre obiettivi: si possono avere design pattern di tipo “classe” o di tipo “oggetto”, ognuno dei quali può risolvere problemi creazionali, strutturali e comportamentali.
Quelli di tipo classe utilizzano l’ereditarietà per collegare tra loro le classi, una connessione forte creata a tempo di compilazione, mentre quelli di tipo oggetto sfruttano la composizione, creano dipendenze dichiarando oggetti dentro altri oggetti, connessioni deboli create a runtime ma che garantiscono una maggiore flessibilità rispetto all’ereditarietà.
I design pattern creazionali sono utili quando l’obiettivo principale è creare nuove classi, quelli strutturali risolvono problemi strutturali delle classi, mentre quelli comportamentali vengono utilizzati per modificare il comportamento delle classi.

BestClient utilizza nel suo funzionamento due design pattern principali: uno Strategy e un Adapter, il primo di tipo oggetto e comportamentale, il secondo di tipo oggetto e strutturale.

5.2. I Princìpi SOLID

Ogni design pattern soddisfa almeno uno dei cinque princìpi per lo sviluppo di software di qualità, noti anche come princìpi SOLID:

- **Single Responsibility Principle**: ogni classe deve avere una sola ragione per cambiare, deve avere una sola funzionalità e deve dipendere da un solo attore, in modo che se la funzionalità cambia devo modificare solo quella determinata classe e non le altre.
- **Open Closed Principle**: ogni classe deve essere chiusa rispetto alle modifiche e aperta alle estensioni, in modo da progettare moduli che non devono cambiare, facilitando la gestione dei cambiamenti e riducendo l'accoppiamento tra le classi.
- **Liskov Substitution Principle**: quando si costruisce una gerarchia tra le classi, la sottoclasse deve avere lo stesso comportamento della classe padre, ovvero deve essere in grado di poter implementare tutti i metodi della sua superclasse.
- **Interface Segregation Principle**: una classe deve dipendere unicamente dalle classi che utilizza, evitando di avere grandi interfacce e classi che non implementano tutti i loro metodi.
- **Dependency Inversion Principle**: sia gli oggetti di alto livello sia gli oggetti concreti devono dipendere da astrazioni per garantire la stabilità del software.

5.3. Strategy

Il design pattern Strategy viene implementato attraverso la classe UIManager visibile nel class diagram, e consente di poter cambiare a runtime e in maniera lineare l'elenco di valutazioni della home page, infatti è sufficiente invocare il metodo *setVisualizzatore()* della classe MainActivity per cambiare l'elenco numeri tra quelli disponibili (chiamate in entrata, chiamate in uscita, contatti e "mie valutazioni"). Questo design pattern permette inoltre di rendere più flessibile l'applicazione nel caso in cui si verifichino dei cambiamenti, come l'aggiunta di un altro visualizzatore o la rimozione di uno di questi, rispettando così il principio di Open-Closed per cui una classe deve essere chiusa rispetto alle modifiche e aperta alle estensioni. Infatti, se in futuro si volesse aggiungere un nuovo elenco di valutazioni basterebbe creare una nuova classe apposita che estenda UIManager.

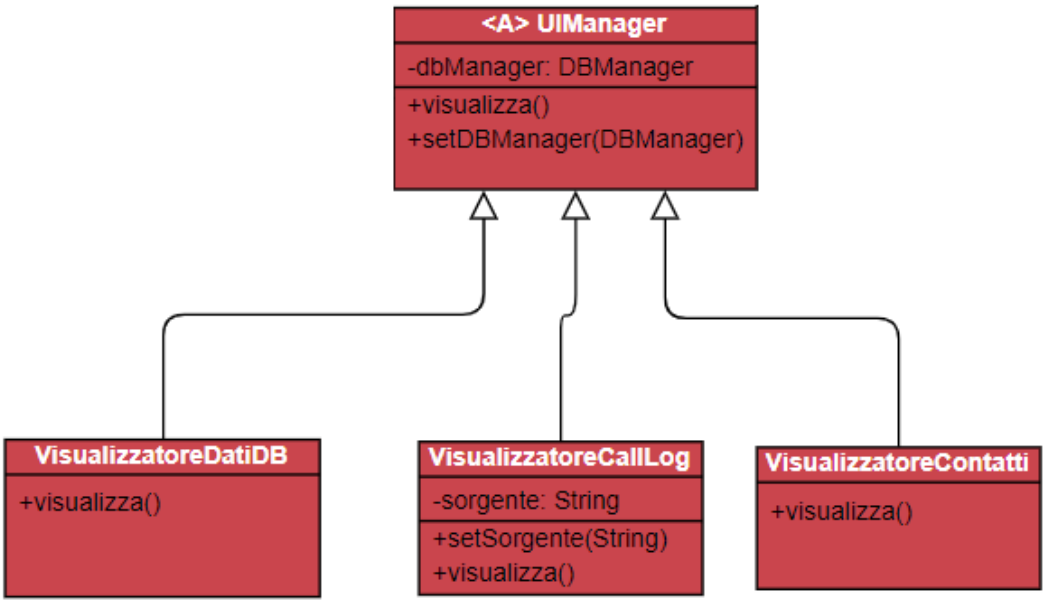

Figura 26: Strategy

5.4. Adapter

Il design pattern Adapter viene utilizzato per risolvere il requisito non funzionale di rendere flessibile l'applicazione nel caso di un cambio di database. L'interfaccia DBManager contiene al suo interno la dichiarazione di tutti i metodi fondamentali che deve avere la classe che la implementa, nel caso di BestClient la classe FirebaseAdapter, e che fa da tramite tra l'applicazione e le funzioni dedicate del database utilizzato. Se si volesse infatti cambiare database basterebbe creare una nuova classe Adapter che implementi DBManager e che contenga i metodi necessari a recuperare tutte le informazioni dal database per passarle all'applicazione. Anche questo design pattern rispetta il principio di Open-Closed come lo Strategy.

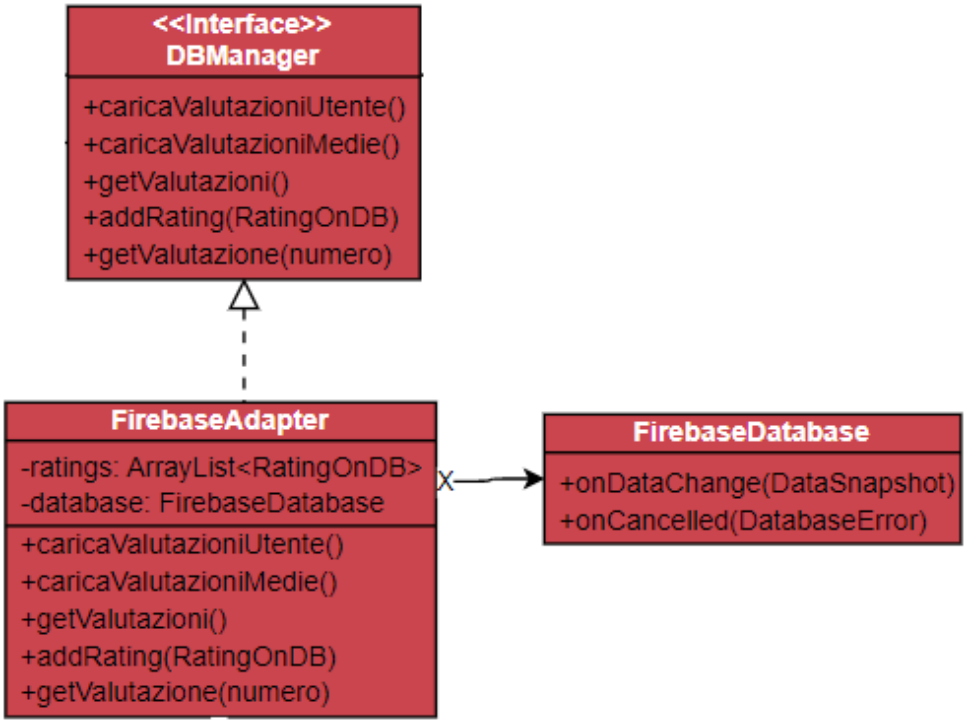

Figura 27: Adapter

5.5. Design pattern “impliciti”

All’interno di BestClient sono presenti alcuni schemi di funzionamento tipici di alcuni dei design pattern più utilizzati, come il Template, il Decorator e l’Observer. Si è scelto di non implementarli concretamente perché alcune caratteristiche dell’applicazione non si adattano perfettamente alla struttura delle classi di questi pattern, ma in linea di principio il meccanismo di funzionamento è molto simile, per questo sono stati soprannominati design pattern “impliciti”.

5.5.1. Template

Il design pattern Template consente di definire gli step di un algoritmo e permette ad alcune sottoclassi di implementare diversamente alcuni di questi passi. È un pattern di tipo classe e comportamentale e si compone di una classe astratta principale in cui il metodo *templateMethod()*, dichiarato come final e pertanto non modificabile da nessuna sottoclasse, contiene l’invocazione ordinata di tutti i metodi che rappresentano gli step dell’algoritmo da implementare, in parte dichiarati anch’essi come final e già implementati, come *doAbsolutelyThis()*, e in parte non implementati, come *primitiveOperation1()* e *primitiveOperation2()*. Il metodo *hook()* rappresenta una funzionalità aggiuntiva o un caso particolare che le sottoclassi possono scegliere di implementare o meno. Ogni classe che estende la principale deve obbligatoriamente implementare le operazioni primitive per poter costruire varianti dell’algoritmo da implementare. Il pattern segue il principio Hollywood, per il quale è la classe astratta a chiamare le implementazioni, infatti durante l’esecuzione viene istanziata una delle classi concrete ma viene assegnata ad un riferimento della superclasse Template, così da poter invocare il metodo *templateMethod()* ed eseguire l’algoritmo step dopo step utilizzando i metodi concreti della sottoclasse e della classe principale.

Il suo principio di funzionamento può essere ricondotto a quello di BestClient nello scenario di inserimento di una valutazione, infatti l'algoritmo potrebbe prevedere diversi step:

1. Selezione del numero da valutare, metodo concreto e uguale per tutte le sottoclassi;
2. Inserimento della valutazione, facoltativo;
3. Inserimento di un commento pubblico, facoltativo;
4. Inserimento di un commento privato, facoltativo;

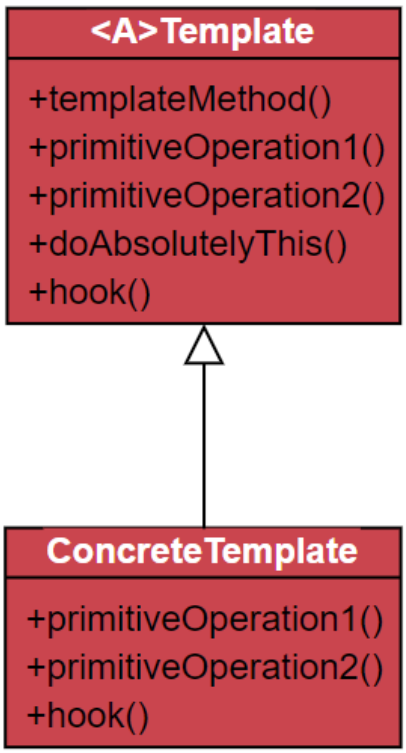

Figura 28: Class Diagram Template

5.5.2. Decorator

Il design pattern Decorator consente di cambiare dinamicamente il comportamento dei metodi di una classe, concependo un oggetto come tanti oggetti uno dentro l'altro, è un pattern di tipo oggetto e strutturale ed è un'alternativa flessibile all'uso delle sottoclassi. Lavora costruendo una classe principale, denominata Component, rappresentante l'oggetto a cui si possono aggiungere i vari attributi, una classe concreta ConcreteComponent che la estende e contiene le funzionalità di base del componente, una classe Decorator che rappresenta una generica funzionalità e che viene estesa dalle classi ConcreteDecorator, contenenti le implementazioni vere e proprie dei metodi da aggiungere. Dal punto di vista software si istanzia prima la classe di base e poi si istanziano i vari componenti aggiuntivi, passando al costruttore l'oggetto precedente. In questo modo si crea una sorta di matriosca per cui, una volta invocato un metodo, questo viene eseguito ricorsivamente fino alla classe base e viene ritornato il risultato finale comprendente tutte le funzionalità aggiunte.

Questo pattern rispetta il principio di Open-Closed, consentendo di estendere una classe senza modificarla.

Il suo schema di funzionamento si può ricondurre a BestClient nell'oggetto RatingOnDB rappresentante la valutazione di un utente. La stessa infatti ha come componente di base il numero di telefono relativo alla persona da valutare, la data in cui avviene la valutazione, e come componenti aggiuntivi il voto numerico, il commento, il flag pubblico o privato, il voto medio, la lista dei commenti di altri utenti e il numero di persone che hanno contribuito alla valutazione media.

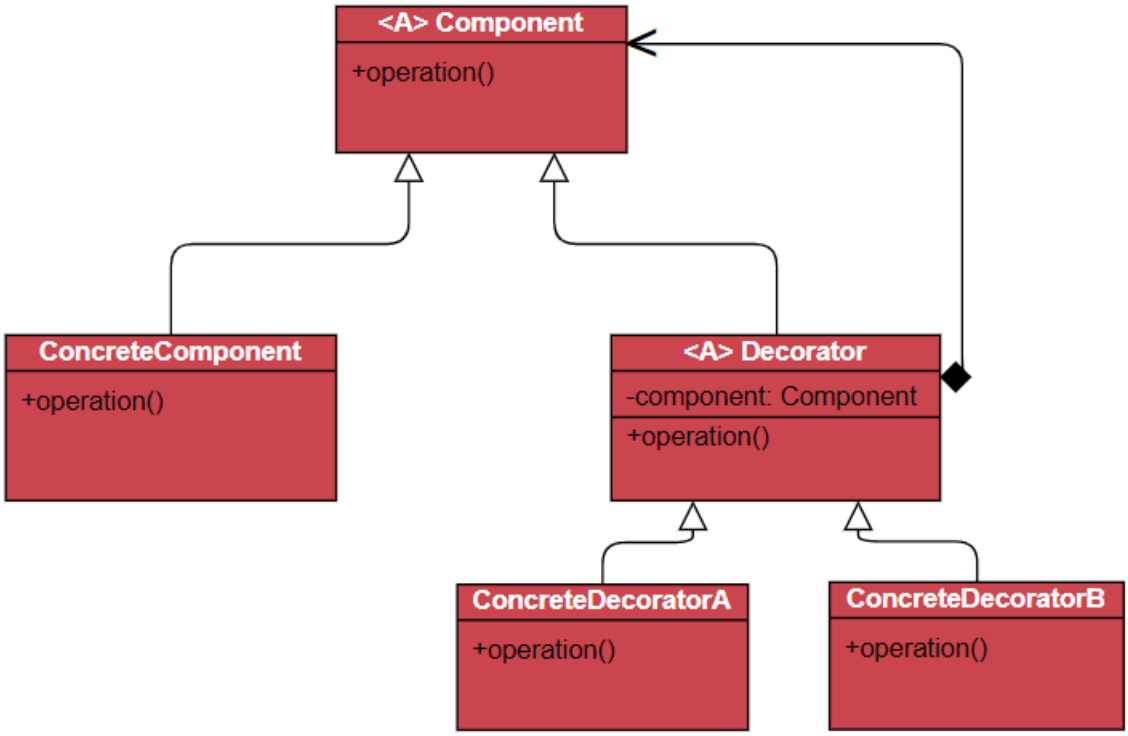

Figura 29: Class Diagram Decorator

5.5.3. Observer

Il design pattern Observer permette agli oggetti di un'applicazione di essere mantenuti e aggiornati da altri oggetti, è tipico dei listener ed è di tipo oggetto e comportamentale. Si compone di due interfacce principali Observable e Observer e di due classi concrete che le implementano: la prima rappresenta l'oggetto responsabile degli aggiornamenti, mentre la seconda è relativa agli oggetti da aggiornare. Ogni Observer deve registrarsi ad un Observable invocando il metodo *registerObserver()* per ricevere gli aggiornamenti, così quando si verifica un cambiamento nella classe Observable viene invocato il metodo *notifyObservers()*, che scorre tutta la lista di Observer registrati e invoca per ognuno il metodo *update()* passandogli i nuovi dati aggiornati. Ogni aggiornamento può essere passato come singoli parametri (meccanismo push), oppure come oggetto intero da cui estrarre i nuovi valori dagli attributi (meccanismo pull). Questo pattern utilizza il principio di poco accoppiamento tra le classi per minimizzare la dipendenza tra gli oggetti e scambiare solo le informazioni necessarie.

All’interno di BestClient il funzionamento di un Observer è implementato dalla classe IncomingReceiver, che si registra al listener del gestore chiamate in attesa che arrivi una chiamata in entrata, così da ricevere il numero di telefono del chiamante e mostrare la notifica all’utente.

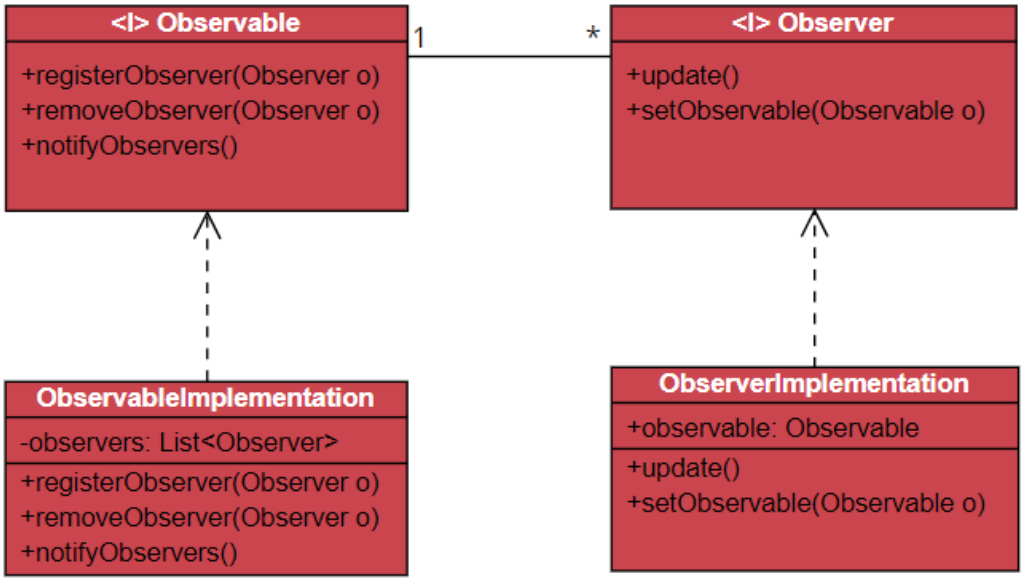

Figura 30: Class Diagram Observer

6. Appendice

6.1. Requisiti funzionali

- L’applicazione deve permettere all’utente di valutare i propri clienti in merito all’interazione che hanno avuto con l’utente stesso.

- Il cliente deve poter inserire le sue valutazioni ricercando per numero telefonico all’interno del registro chiamate e fra i propri contatti.

- Il cliente deve poter modificare le proprie valutazioni nel tempo e deve avere la possibilità di inserire un commento facoltativo che deciderà se tenerlo per sé o renderlo pubblico per gli altri utenti dell’applicazione.

- Alla ricezione di una telefonata il cliente deve poter visualizzare la valutazione data a quel numero telefonico e la valutazione media data dagli altri utenti.

- L’utente deve avere anche la possibilità di ricercare una valutazione specifica all’interno del database inserendo il numero di telefono a cui è associato il voto.

- Un utente deve essere autenticato per utilizzare l’applicazione e ricevere notifiche.

6.2. Requisiti non funzionali

- L’applicazione deve essere estendibile nel caso di un cambio di database.

- L’applicazione deve essere installabile su dispositivi con sistema operativo Android.

- Per sfruttare tutte le funzionalità sono necessari i permessi di lettura del registro chiamate, della rubrica e delle chiamate in arrivo.

6.3. Classificazione dei requisiti

Numero del requisito	Descrizione del requisito	Tipologia	Priorità	Stato del requisito
1	Possibilità di valutare i clienti in merito all'interazione con l'utente	Funzionale	1	Implementato
2	Ricerca numeri di telefono da valutare dal registro chiamate e dalla rubrica	Funzionale	1	Implementato
3	Possibilità di inserire un commento pubblico o privato	Funzionale	2	Implementato
4	Ricezione di una notifica contenente la valutazione personale e collettiva	Funzionale	1	Implementato
5	Ricerca di una valutazione specifica inserendo il numero di telefono relativo	Funzionale	3	Implementato
6	Un utente deve essere autenticato per poter utilizzare l'applicazione	Funzionale	1	Implementato
7	Flessibilità in caso di cambio di database	Non Funzionale	3	Strutturato per prossima release
8	Applicazione disponibile sui sistemi Android	Non Funzionale	1	Implementato
9	Accesso al registro chiamate, alla rubrica e alle chiamate in entrata	Non Funzionale	1	Implementato, a scelta dell'utente

Tutti i requisiti sono stati implementati nella release corrente dell'applicazione, l'unica eccezione è il requisito numero 7 che prevede il refactoring del codice e della struttura dell'applicazione, già impostato per essere inserito nella prossima release.

L'ordine di priorità dei requisiti va da 1 a 3 per indicare su quali aspetti ci siamo focalizzati maggiormente durante lo sviluppo dell'applicazione.

La priorità 1 indica un requisito più importante ed è stata assegnata alle funzionalità principali e caratteristiche di BestClient, tra cui la possibilità di valutare un cliente attraverso il suo numero di telefono, la necessità di accedere al registro chiamate e alla rubrica contatti per poter sfruttare tutte le funzioni dell'applicazione, la registrazione e la ricezione della notifica all'arrivo di una chiamata.

La priorità 2 è stata assegnata alla possibilità di inserire un commento pubblico o privato che esprime un giudizio esplicativo sul voto assegnato. Inizialmente si era pensato anche di poter utilizzare il commento privato come uno strumento temporaneo per ricordarsi ad esempio il numero del tavolo in cui era seduto il cliente, così da non dover salvare il suo numero di telefono ed esprimere la valutazione vera e propria una volta che il cliente ha lasciato il locale.

La priorità 3 indica i requisiti meno importanti su cui ci si è focalizzati dopo aver implementato quelli principali, come la possibilità di ricercare una valutazione specifica tramite una barra di ricerca inserendo il numero di telefono o il nome del contatto relativo, oppure rendere l'applicazione estendibile in caso di cambio di database, supponendo che questo non cambi almeno nel breve periodo.

6.4. Analytical Hieralchical Process

Con Analytical Hierarchical Process si intende il processo con cui si riesce ad esprimere un valore di priorità per ogni requisito. Per ottenere questo risultato si mettono i requisiti in tabella, si confrontano uno ad uno e si esprime un valore numerico di importanza di uno rispetto all'altro, dopodiché si sommano tutte le colonne, si divide ogni casella per il risultato appena ottenuto e infine si somma ogni riga. Questo valore viene diviso per il numero di requisiti e moltiplicato per cento in modo da ottenere un valore che esprime la percentuale di importanza di quel requisito rispetto agli altri.

Come primo passo assegniamo quindi un valore di priorità confrontando ogni requisito rispetto agli altri e sommiamo i valori di ogni colonna. Si può notare che in questo caso valori più alti esprimono un grado di priorità maggiore e che la tabella risulta simmetrica.

	Req1	Req2	Req3	Req4	Req5	Req6	Req7	Req8	Req9
Req1	1	2	4	2	4	1	4	1	2
Req2	1/2	1	3	1	4	1	4	1	1
Req3	1/4	1/3	1	1/4	2	1	2	1	1/3
Req4	1/2	1	4	1	4	1	4	1	2
Req5	1/4	1/4	1/2	1/4	1	1	3	1	2
Req6	1	1	1	1	1	1	1	1	1
Req7	1/4	1/4	1/2	1/4	1/3	1	1	1	1/4
Req8	1	1	1	1	1	1	1	1	1
Req9	1/2	1	3	1/2	1/2	1	4	1	1
TOT	5.25	7.83	18	7.25	17.83	9	24	9	10.58

Ora dividiamo ogni cella per la somma della colonna corrispondente e sommiamo per riga.

	Req1	Req2	Req3	Req4	Req5	Req6	Req7	Req8	Req9	TOT
Req1	0.19	0.255	0.22	0.275	0.224	0.11	0.17	0.11	0.19	1.744
Req2	0.095	0.13	0.17	0.14	0.224	0.11	0.17	0.11	0.094	1.243
Req3	0.047	0.042	0.055	0.034	0.112	0.11	0.083	0.11	0.031	0.624
Req4	0.095	0.13	0.22	0.14	0.224	0.11	0.17	0.11	0.19	1.389
Req5	0.047	0.032	0.027	0.034	0.056	0.11	0.014	0.11	0.19	0.62
Req6	0.19	0.13	0.055	0.14	0.056	0.11	0.042	0.11	0.094	0.927
Req7	0.047	0.032	0.027	0.034	0.019	0.11	0.042	0.11	0.024	0.445
Req8	0.19	0.13	0.055	0.14	0.056	0.11	0.042	0.11	0.094	0.927
Req9	0.095	0.13	0.17	0.07	0.028	0.11	0.17	0.11	0.094	0.977

Il passo finale è quello che ci permette di ottenere la priorità percentuale di ogni requisito rispetto agli altri. Si può notare che i requisiti elencati in precedenza che avevano la massima priorità ora hanno ancora i valori percentuali più alti. Tra tutti spicca con un valore del 20% il requisito che esprime lo scopo principale di BestClient, ovvero la possibilità di valutare i clienti, al secondo posto troviamo la ricezione della notifica con i voti relativi al chiamante, mentre al terzo posto vediamo la ricerca dei numeri di telefono da valutare tramite il registro chiamate e la rubrica contatti.

Requisito	Priorità %
Req1	20%
Req2	14%
Req3	7%
Req4	16%
Req5	7%
Req6	10%
Req7	5%
Req8	10%
Req9	11%

7. Conclusioni e futuri sviluppi

È stato un piacere realizzare un'applicazione commissionata da una persona che, partendo da un'idea per risolvere un problema di gestione dei suoi locali, è arrivato ad una soluzione che gli permetta di svolgere la sua attività al meglio.

È la prima applicazione che sviluppo per Android e ho avuto modo di imparare un nuovo ambiente di programmazione utilizzato da moltissimi sviluppatori in tutto il mondo, ma soprattutto ho iniziato ad avere un'idea di quali sono le fasi, le prospettive dello sviluppo di un software e l'interazione con chi, partendo da un'idea, riesce ad arrivare alla sua concreta realizzazione attraverso un'applicazione.

Nelle prossime release verrà implementato il requisito non funzionale di flessibilità nel caso di cambio di database e la verifica dell'account tramite un'e-mail di conferma. Sarà necessario anche controllare la veridicità della partita IVA inserita, aspetto non ancora implementato nella versione beta dell'applicazione. Al momento BestClient è in uso da parte di una decina di ristoratori bolognesi in modo da poter avere un feedback sugli aspetti da modificare o da aggiungere prima di rilasciare la versione vera e propria dell'applicazione. È stata pensata anche una versione iOS ma ci sono difficoltà nella gestione del registro chiamate a causa delle policy Apple, pertanto risulterebbe un'applicazione incompleta che permetterebbe di valutare solo i contatti presenti in rubrica oppure di valutare inserendo ogni volta il numero di telefono nella barra di ricerca, aspetto non trascurabile e che porterebbe ad un uso sicuramente minore dell'applicazione.

8. Bibliografia

- Figura 1: Il potere delle recensioni, fonte: Capterra, https://www.capterra.it/blog/1808/studio-recensioni-online-clienti
- Figura 2: Le motivazioni delle recensioni, fonte: Capterra, https://www.capterra.it/blog/1808/studio-recensioni-online-clienti
- Logo Android Studio: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.stefanoivancich.com%2F%3Fp%3D1300&psig=AOvVaw3yKdojAiR2gf9ChohxBQrQ&ust=1649935112115000&source=images&cd=vfe&ved=0CAoQjRxqFwoTCNiOnrf1kPcCFQAAAAAdAAAAABAm
- Logo Firebase: https://www.google.com/url?sa=i&url=https%3A%2F%2Fit.m.wikipedia.org%2Fwiki%2FFile%3AFirebase_Logo.svg&psig=AOvVaw3DbZxfYl8eU3expwgva4Tl&ust=1649935526602000&source=images&cd=vfe&ved=0CAoQjRxqFwoTCMDervr2kPcCFQAAAAAdAAAAABAD
- Logo Github: https://www.google.com/url?sa=i&url=https%3A%2F%2Flogos-world.net%2Fgithub-logo%2F&psig=AOvVaw20KlgmTYKzSF-erS8mHS3e&ust=1650454361698000&source=images&cd=vfe&ved=0CAkQjRxqFwoTCJDS0qSRoPcCFQAAAAAdAAAAABAD
- Logo WayScript: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.owler.com%2Fcompany%2Fwayscript&psig=AOvVaw2YZAxEWJrg74bYy9K4azSx&ust=1650455855469000&source=images&cd=vfe&ved=0CAkQjRxqFwoTCLC8h7WRoPcCFQAAAAAdAAAAABAD
- Diagrammi realizzati con Visual Paradigm Online, https://online.visual-paradigm.com/