Documentazione progetto

Brumana Matteo, Foster Timothy James, Rota Matteo
matricole n. 1053588, 1053643, 1053065

Università degli Studi di Bergamo

Laurea Magistrale in Ingegneria Informatica

Corso di Informatica III

Modulo di Progettazione e Algoritmi (6 CFU)

Anno Accademico 2021/2022

INDICE

ITERAZIONE 0

1.1 ANALISI DEL CONTESTO

Per parrucchieri ed estetisti l'agenda cartacea rappresenta ancora lo strumento più diffuso per gestire gli appuntamenti; tuttavia, carta e penna sono talmente semplici e comodi che risulta difficile accorgersi degli svantaggi che portano, tra cui figurano volatilità, mancanza di standard e necessità di averli con sé per poterli utilizzare.

Questo progetto si pone l'obiettivo di creare un'agenda virtuale facile da utilizzare quanto quella cartacea ma in grado di eliminarne i problemi e di ampliarne i vantaggi, mirando a migliorare la gestione del lavoro di un qualsiasi salone, ad esempio tramite il suggerimento di slot temporali per i prossimi appuntamenti, e di aumentare la fidelizzazione dei clienti dello stesso, ad esempio tramite l'emissione di buoni sconto di compleanno o l'invio di reminder in vista degli appuntamenti fissati.

Il risultato degli sforzi progettuali sarà un prodotto versatile, facilmente configurabile, ancor più facilmente utilizzabile e soprattutto basato il più possibile su tecnologie cloud, che elimineranno il costo d'acquisto e manutenzione di hardware locale e renderanno il sistema sempre raggiungibile senza alcuno sforzo economico o di configurazione.

1.2 REQUISITI

Un sistema pensato per titolari di attività per la cura della persona dovrà:

1. gestire gli appuntamenti con i clienti;
    1.1. dovrà essere possibile creare/modificare/eliminare appuntamenti;
        1.1.1. durante la creazione/modifica di un appuntamento, dovrà essere possibile specificare chi è il cliente interessato;
            1.1.1.1. durante la specifica del cliente interessato dall'appuntamento, dovrà essere possibile inserire dati su nuovi clienti a sistema [punto 2.1];
        1.1.2. durante la creazione/modifica di un appuntamento, dovrà essere possibile specificare i servizi richiesti dal cliente e far decidere al sistema la durata dell'appuntamento;
            1.1.2.1. il sistema dovrà conoscere le specifiche dei vari servizi erogabili;
    1.2. dovrà essere possibile visionare gli appuntamenti creati su un calendario;
    1.3. il sistema dovrà inviare automaticamente via SMS ai clienti un reminder relativo al loro appuntamento 24h prima dello stesso;
2. gestire le anagrafiche dei clienti;
    2.1. dovrà essere possibile inserire/modificare/eliminare dati sui clienti;

2.2. dovrà essere possibile consultare i dati registrati sui clienti;

3. gestire i buoni sconto emessi in occasione dei compleanni dei clienti;
   3.1. il sistema dovrà occuparsi di generare automaticamente buoni sconto in occasione dei compleanni dei clienti e di inviarli via SMS agli stessi al momento della generazione;
   3.2. il sistema dovrà occuparsi di eliminare automaticamente buoni sconto quando la loro data di scadenza risulta superata;

1.3 STUDIO DI FATTIBILITÀ

Dopo numerose discussioni, si è deciso di implementare il sistema descritto nei requisiti mediante un'app Android, supportandola con un database cloud e funzioni cloud.
Per realizzare tutto questo, sono state selezionate le seguenti tecnologie:

- Database cloud:
  La scelta, data la familiarità con esso acquisita durante il corso di Tecnologie Cloud e Mobile, è ricaduta su MongoDB Atlas, database distribuito totalmente in cloud realizzato con tecnologia MongoDB (NoSQL, document-based).
  La particolare struttura di questo database comporta dei requisiti maggiori per il livello di applicazione, ma consente di distribuire dati e processi su più server, rendendolo scalabili in maniera quasi illimitata e garantendo disponibilità e flessibilità dei dati;
- Connettore tra app Android e database cloud:
  Il collegamento tra applicazione e database viene comodamente offerto dall'API Realm di MongoDB per ambiente Java, che offre l'accesso a tutte le operazioni effettuabili sul database mediante oggetti facilmente manipolabili. La scelta in questo ambito è dunque ricaduta su questa tecnologia.
- Ambiente di esecuzione in cloud:
  Per questo servizio è stato scelto Amazon Web Services, visto anch'esso durante il corso di Tecnologie Cloud e Mobile, e in particolare la sua interfaccia Lambda Functions, che permette di sfruttarne la potenza computazionale eseguendo codice scritto in diversi linguaggi di programmazione in ambiente preconfigurato.
- Connettori tra ambiente di esecuzione cloud e database cloud:
  Il driver MongoDB nativo per Node.js, su cui è ricaduta la scelta, è una dipendenza che consente alle applicazioni che lo sfruttano di interagire con MongoDB Atlas

mediante oggetti che forniscono un punto di contatto con le API di MongoDB e permettono di eseguire numerosi tipi di operazione sui dati presenti nel database.

- Servizio di notifica SMS:
  È stata scelta Twilio, REST API servita su HTTPS che permette l'invio di messaggi SMS;

1.4 TOOLCHAIN

Viste le considerazioni riportate nello studio di fattibilità, è stata determinata la seguente toolchain per il progetto:

Obiettivo	*Strumenti*
Progettazione	Astah UML
	Carta e penna
	Word
Sviluppo codice	Java (per app Android)
	Node.js (per Lambda Functions)
Analisi del codice	JUnit (dinamica per app Android)
	MetricsReloaded (statica per app Android)
	Test dinamici manuali (per Lambda Functions)
Distribuzione e versionamento	GitHub
Documentazione	Word

1.5 CASI D'USO

UC1: Gestione appuntamenti

Descrizione: il titolare deve poter essere in grado di gestire gli appuntamenti fissati con i clienti: questo comprende avere la possibilità di inserire, modificare o eliminare appuntamenti e di poterli visionare su un calendario. Il sistema deve inviare un reminder ai clienti aventi un appuntamento 24h prima dello stesso.

Attori coinvolti: titolare, trigger temporale.

SubCases:

- UC1.1: Creazione appuntamenti;
- UC1.2: Visualizzazione appuntamenti;
- UC1.3: Eliminazione appuntamenti;
- UC1.4: Modifica appuntamenti;
- UC1.5: Invio automatico reminder appuntamenti;

UC2: Gestione clienti

Descrizione: il titolare deve avere a disposizione un archivio contenente dati sui propri clienti: questo comprende avere la possibilità di inserire, modificare o eliminare dati anagrafici e di poterli visionare.

Attori coinvolti: titolare.

SubCases:

- UC2.1: Creazione anagrafiche clienti;
- UC2.2: Visualizzazione anagrafiche clienti;
- UC2.3: Eliminazione anagrafiche clienti;
- UC2.4: Modifica anagrafiche clienti;

UC3: Gestione buoni sconto compleanno

Descrizione: il sistema deve generare e inviare buoni sconto in occasione dei compleanni dei clienti.

Attori coinvolti: trigger temporale.

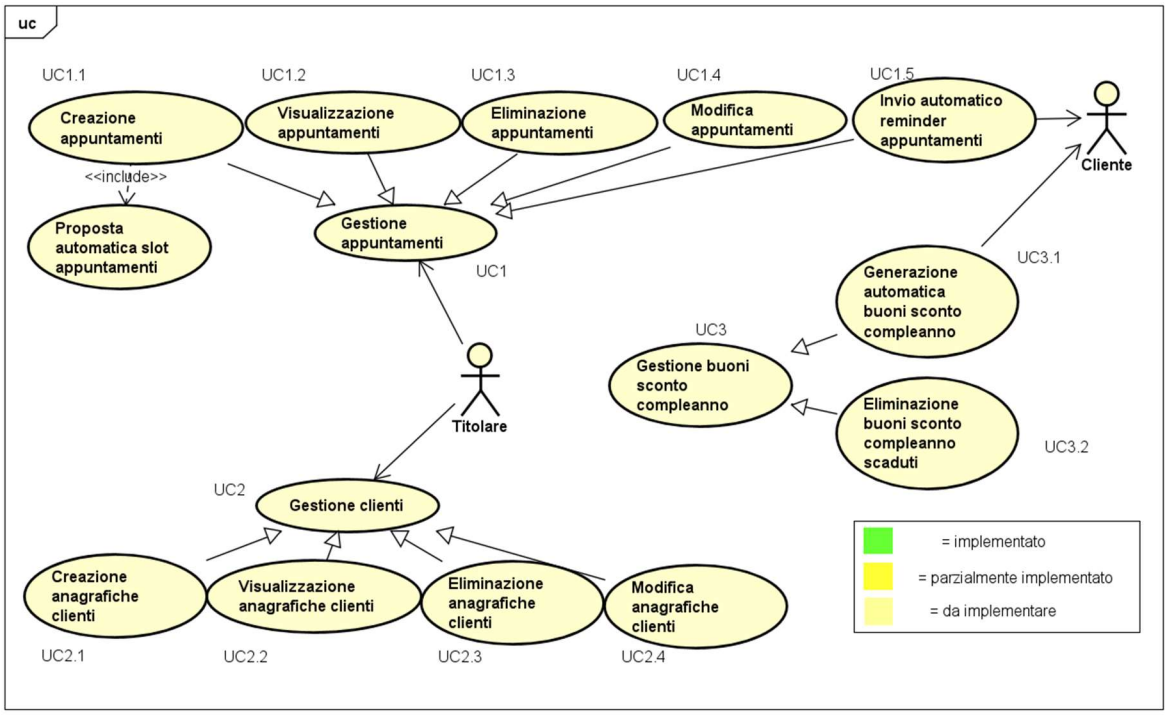
SubCases:

- UC3.1: Generazione automatica buoni sconto compleanno;
- UC3.2: Eliminazione automatica buoni sconto compleanno scaduti

Una volta individuati i casi d'uso, questi sono stati ordinati per priorità di implementazione come segue:

Codice	*Titolo*
UC1	Gestione appuntamenti
UC3	Gestione buoni sconto compleanno
UC2	Gestione clienti

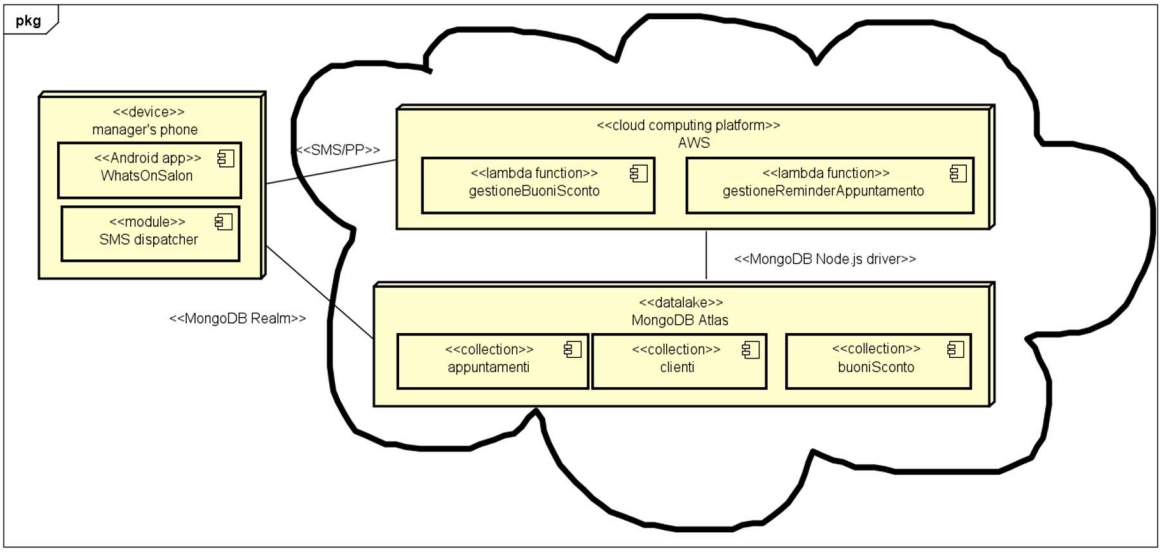
I casi d'uso individuati sono inoltre stati mappati su un diagramma UML dei casi d'uso:



1.6 ARCHITETTURA HARDWARE

Sulla base di quanto discusso in precedenza, è stata elaborata una prima bozza di possibile architettura del sistema ed è stata mappata su un diagramma UML, più precisamente un deployment diagram.

La nuvola presente nel diagramma, ovviamente, rappresenta il cloud e le connessioni tra i vari dispositivi hardware esplicitano il tipo di tecnologia utilizzata per implementarle.

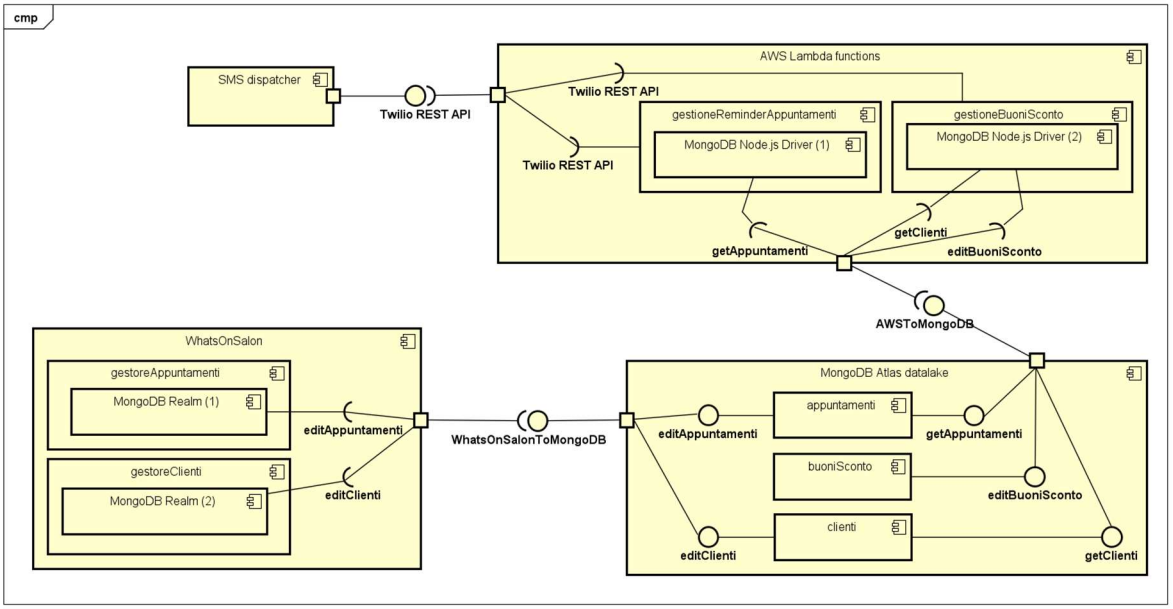


ITERAZIONE 1

2.1 ARCHITETTURA SOFTWARE

Basandosi sul lavoro svolto nell'iterazione precedente, è stato possibile cominciare a progettare l'architettura del software, riassunta in questo diagramma UML delle componenti.

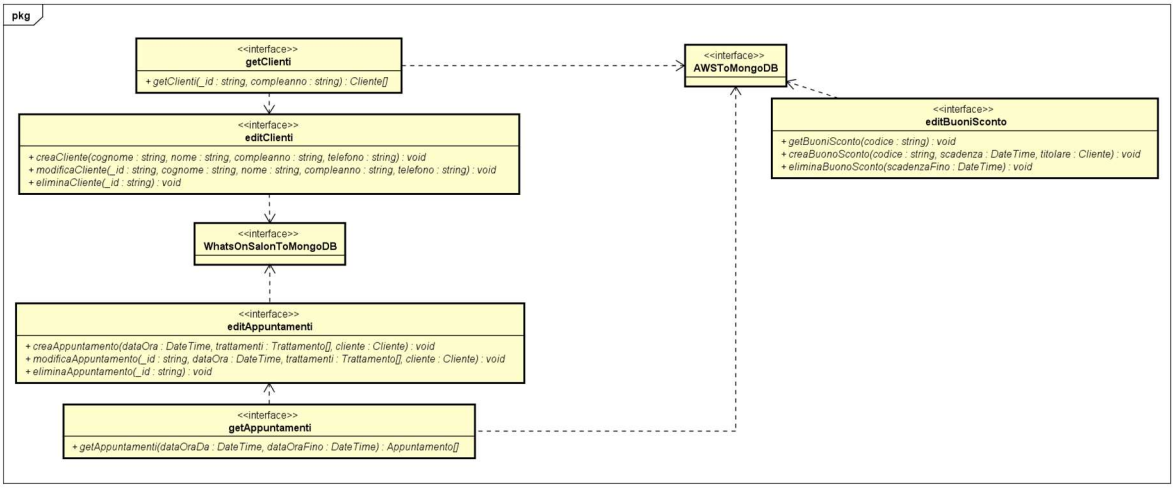
Il diagramma ha lo scopo di rappresentare la struttura interna del sistema software modellato in termini delle sue componenti principali e delle relazioni fra esse. Per facilitarne la comprensione, in esso è stato fatto largo uso di porte e delegation connectors e dove era stato previsto l'utilizzo di software preesistente questo è stato aggiunto direttamente come subcomponent (es. MongoDB Realm) o interfaccia (es. Twilio REST API).



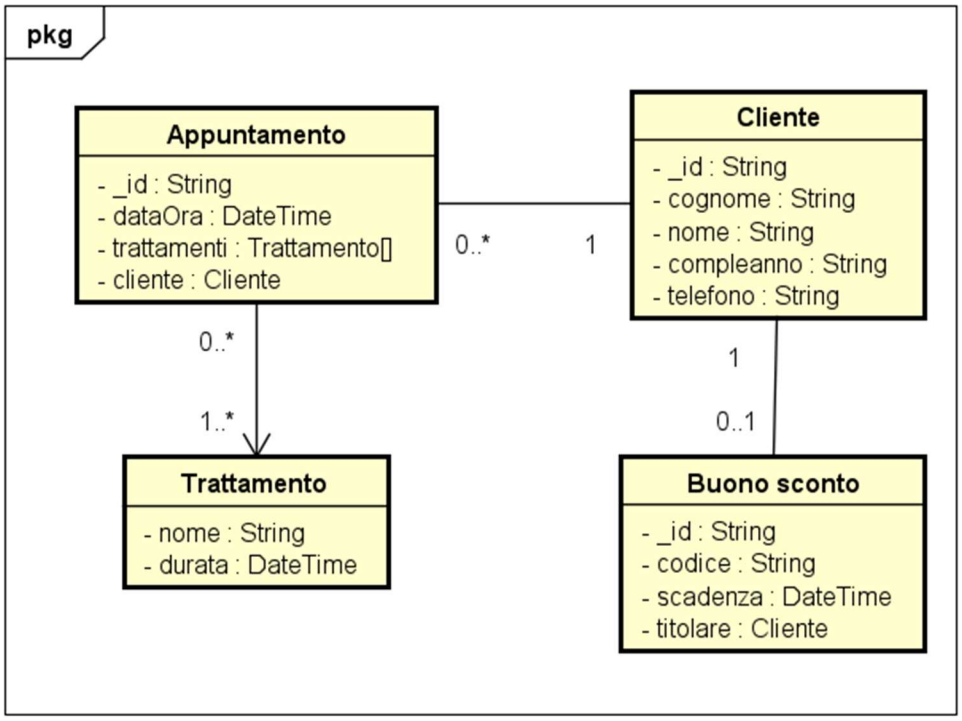
Le interfacce user-defined presenti nel component diagram sono esplicitate in quest'altro schema UML, che prende il nome di interface diagram.

Come si può osservare, esistono relazioni di ereditarietà tra le interfacce: ogni connessione tra component o subcomponent (visibile nel component model) è infatti realizzata da un'interfaccia che include dentro di sé tutte le interfacce richieste/offerte dalle parti in gioco.

I metodi specificati all'interno di ogni interfaccia nello schema sono astrazioni di quella che diverrà la reale implementazione durante le fasi successive di sviluppo.



I tipi di dato user-defined presenti nell'interface diagram, che sono anche quelli che popoleranno il database, e le relazioni che intercorrono tra essi sono esplicitati in questo data diagram che, data la particolare architettura venuta a configurarsi in questo caso, è equivalente e funge anche da data transfer model.

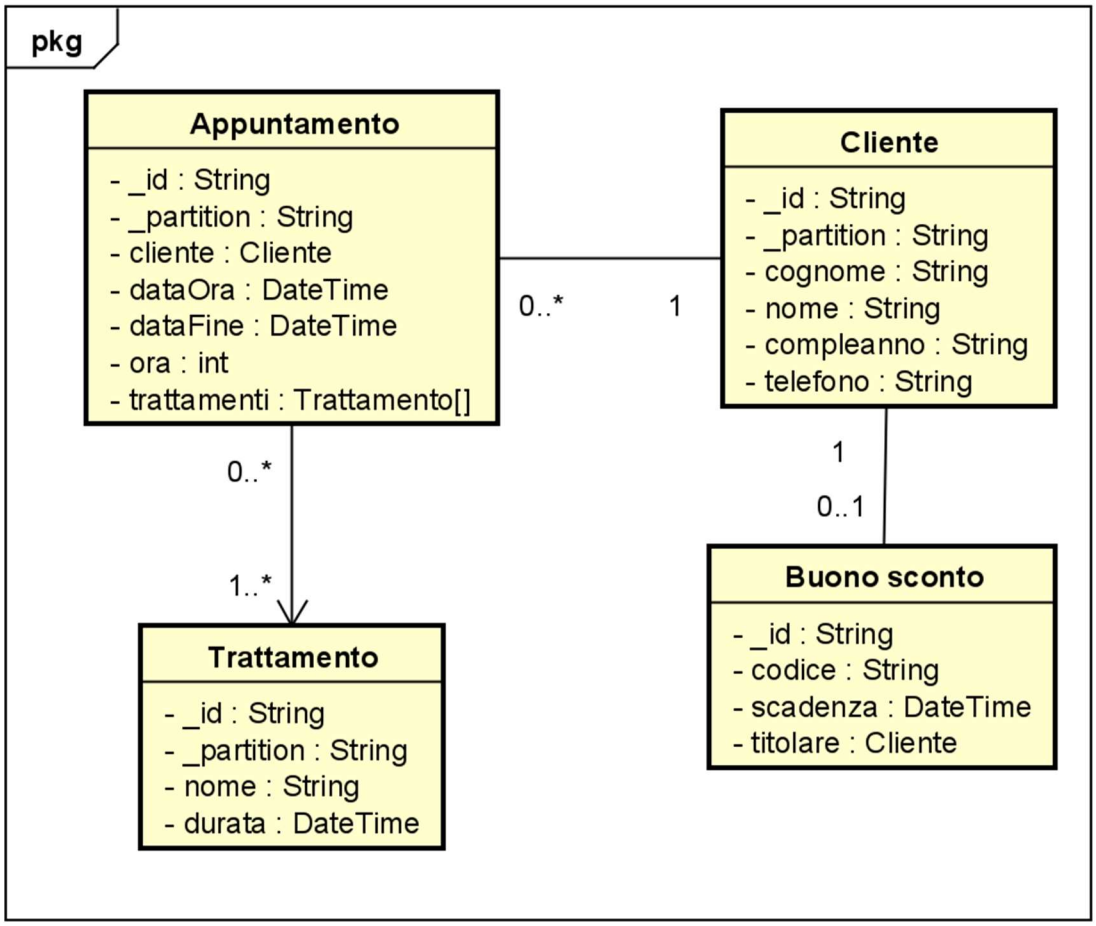


ITERAZIONE 2

3.1 INTRODUZIONE

Durante la preparazione per l'implementazione dei casi d'uso descritti nel paragrafo successivo si è resa necessaria una variazione all'interno del data model, causata dalla necessità di Realm di avere un campo _partition in ogni documento nel database per poterlo recuperare dall'app, dalla comodità di avere un campo dataFine per ogni Appuntamento e dall'inaspettata necessità di dover avere l'orario di inizio di ogni Appuntamento sotto forma di intero per poter effettuare un ordinamento tra Appuntamenti.

Qui di seguito è riportato il data model che caratterizzerà l'elaborato da qui in avanti.

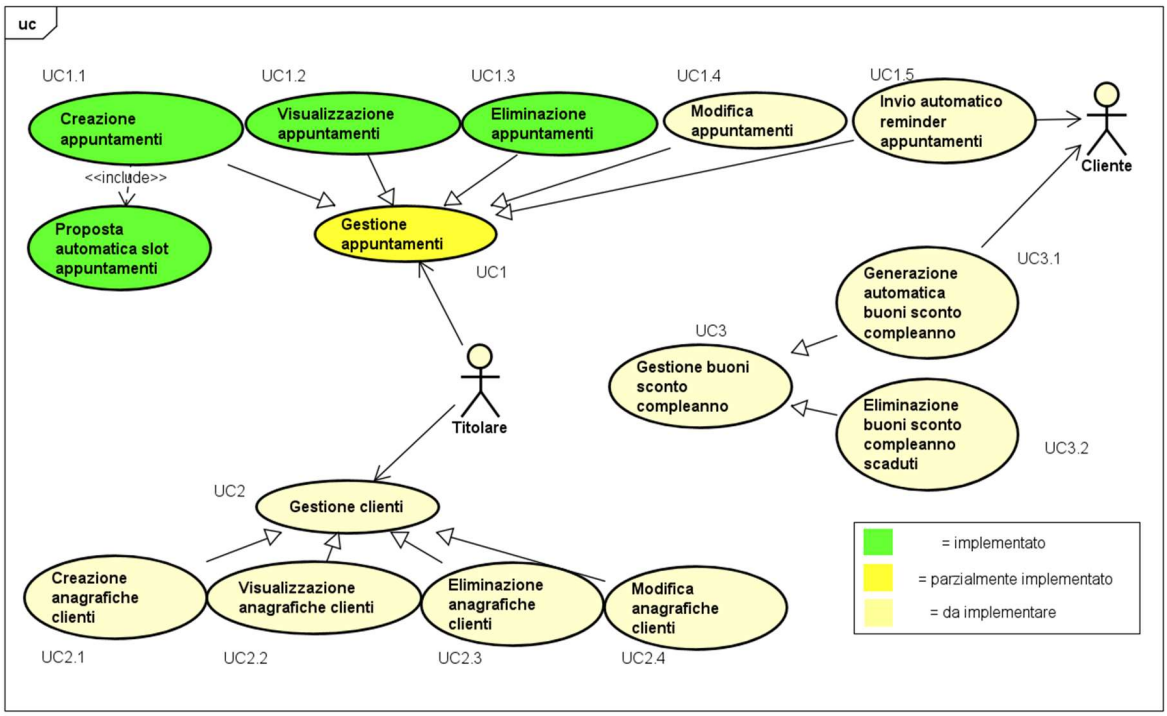


3.2 CASI D'USO SELEZIONATI

Nella seconda iterazione si è deciso di implementare i seguenti casi d'uso:

- UC1: Gestione appuntamenti (astratta):
    - UC1.1: Creazione appuntamenti;
    - UC1.2: Visualizzazione appuntamenti;
    - UC1.3: Eliminazione appuntamenti;

Dal diagramma dei casi d'uso sottostante è possibile analizzare l'avanzamento del progetto facendo riferimento alla legenda in basso a destra.



UC1.1: Creazione appuntamenti

Descrizione: il titolare deve poter essere in grado di fissare appuntamenti sul "calendario virtuale" fornito dall'app, dettagliando per ogni appuntamento cliente interessato (che deve essere inseribile a sistema sul momento, nel caso in cui non fosse già inserito), trattamenti previsti e data e ora nelle quali l'appuntamento è fissato, specificando queste ultime manualmente oppure scegliendole tra le proposte fornite dal sistema.

Attori coinvolti: titolare, algoritmo di proposta automatica slot appuntamenti.

Trigger: richiesta di creazione appuntamento previa specifica dei dati ad esso relativi?.

Postcondizione: appuntamento avente i dati specificati presente sull'apposito database.

Procedimento: 1. il titolare apre l'interfaccia di creazione appuntamenti;

2. il titolare seleziona cliente interessato (eventualmente inserendone l'anagrafica a sistema, nel caso in cui questo non fosse già registrato), i trattamenti previsti

e la data e l'ora in cui l'appuntamento è previsto, selezionandole manualmente o scegliendo una delle proposte provenienti dall'algoritmo di proposta;

3. il titolare richiede la creazione dell'appuntamento appena dettagliato;
4. il titolare viene riportato alla vista del calendario appuntamenti;

UC1.2: Visualizzazione appuntamenti

Descrizione: il titolare deve poter essere in grado di visualizzare in un formato agile tutti gli appuntamenti relativi a un giorno da lui indicato.

Attori coinvolti: titolare.

Trigger: selezione data.

Postcondizione: stampa a video appuntamenti previsti per la data selezionata in un formato facilmente interpretabile.

Procedimento: 1. il titolare seleziona una data in un'interfaccia stile calendario;

2. l'interfaccia viene riempita con gli appuntamenti previsti per la data selezionata, in un formato facilmente interpretabile;

UC1.3: Eliminazione appuntamenti

Descrizione: il titolare deve poter essere in grado di eliminare appuntamenti fissati in precedenza.

Attori coinvolti: titolare.

Trigger: richiesta di eliminazione di uno o più appuntamenti precedentemente selezionati.

Postcondizione: appuntamenti di cui è stata richiesta l'eliminazione non più presenti nel database.

Procedimento: 1. *Procedimento UC1.2* + checkbox di selezione per ogni appuntamento;

2. il titolare seleziona gli appuntamenti che intende eliminare;

3. il titolare richiede l'eliminazione degli appuntamenti selezionati;
4. il titolare viene riportato alla vista del calendario appuntamenti;

3.3 PSEUDOCODICE

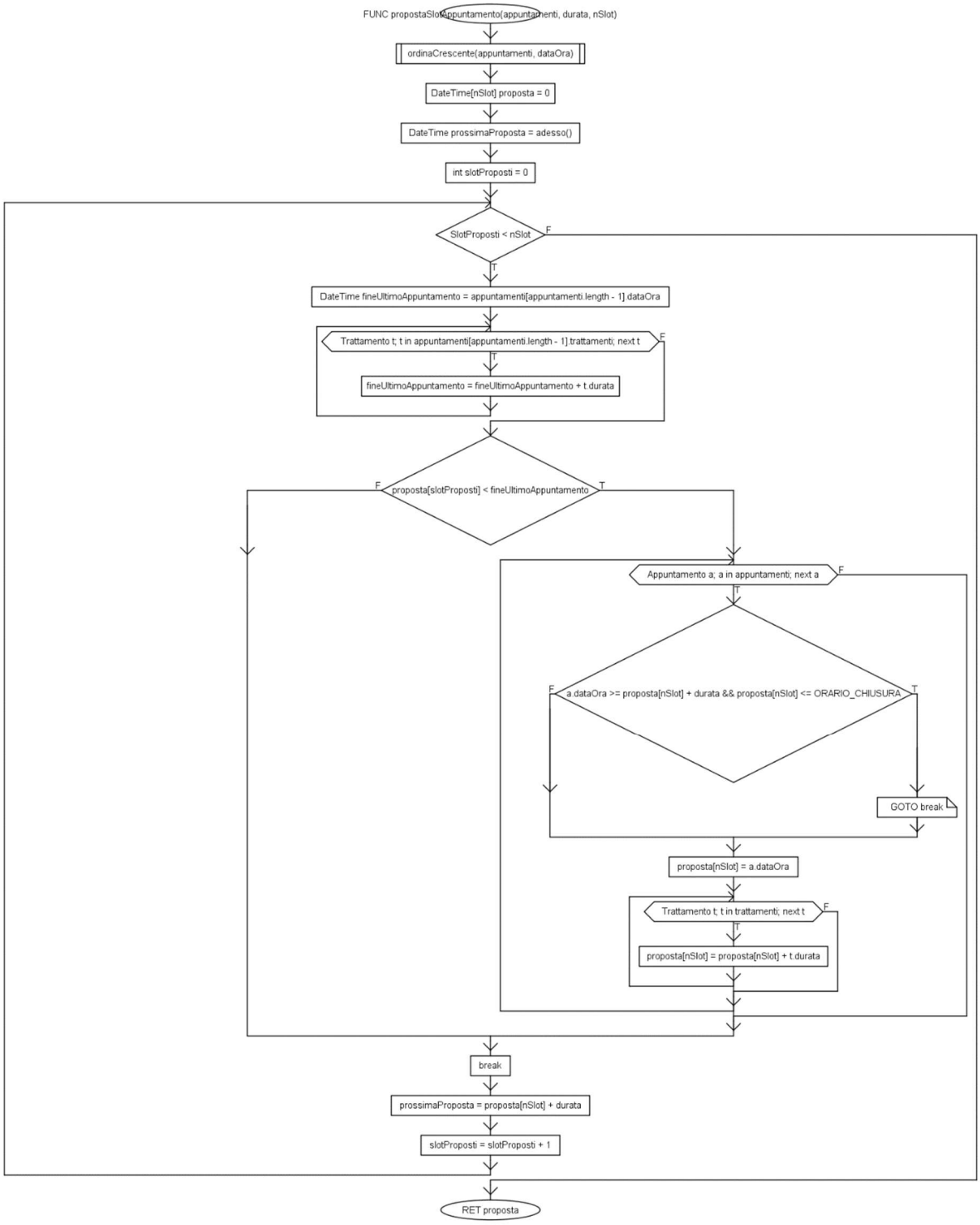
UC1.1: Creazione appuntamenti

```
in: elenco appuntamenti futuri, durata appuntamento da fissare in minuti, numero slot da proporre
out: nSlot proposte di slot per l'appuntamento da fissare
propostaSlotAppuntamento(Appuntamento[] appuntamenti, int durata, int nSlot) -> DateTime[]
    ordinaCrescente(appuntamenti, dataOra) <-- ordino appuntamenti per prossimità


    DateTime[nSlot] proposta <- ∅ <-- array che conterrà le proposte
    DateTime prossimaProposta <- adesso() <-- marker che esclude la proposta multipla dello stesso slot
    int slotProposti <- 0

    while slotProposti < nSlot
        proposta[slotProposti] <- prossimaProposta <-- adesso() nella prima iterazione

        calcolo la fine dell'ultimo appuntamento futuro per capire se sto già proponendo oltre di essa e posso dunque
        evitare la verifica di compatibilità tra la proposta corrente e gli appuntamenti futuri
        DateTime fineUltimoAppuntamento <- appuntamenti[appuntamenti.length - 1].dataOra
        for Trattamento t in appuntamenti[appuntamenti.length - 1].trattamenti
                fineUltimoAppuntamento <- fineUltimoAppuntamento + t.durata

        if proposta[slotProposti] < fineUltimoAppuntamento
          for Appuntamento a in appuntamenti
              verifica che la proposta attuale non interferisca con l'appuntamento successivo
              if (a.dataOra >= proposta[nSlot] + durata && proposta[nSlot] <= ORARIO_CHIUSURA)
                  break <-- se non interferisce, la lascia nell'array delle proposte

              nel caso interferisse, la setta alla fine dell'appuntamento successivo e riprova
              proposta[nSlot] <- a.dataOra
              for Trattamento t in a.trattamenti
                  proposta[nSlot] <- proposta[nSlot] + t.durata

        incrementa il numero di slot proposti e setta come prossima proposta la fine di questo appuntamento se fosse fissato
        alla proposta appena calcolata
        prossimaProposta <- proposta[nSlot] + durata
        slotProposti <- slotProposti + 1

    return proposta
```

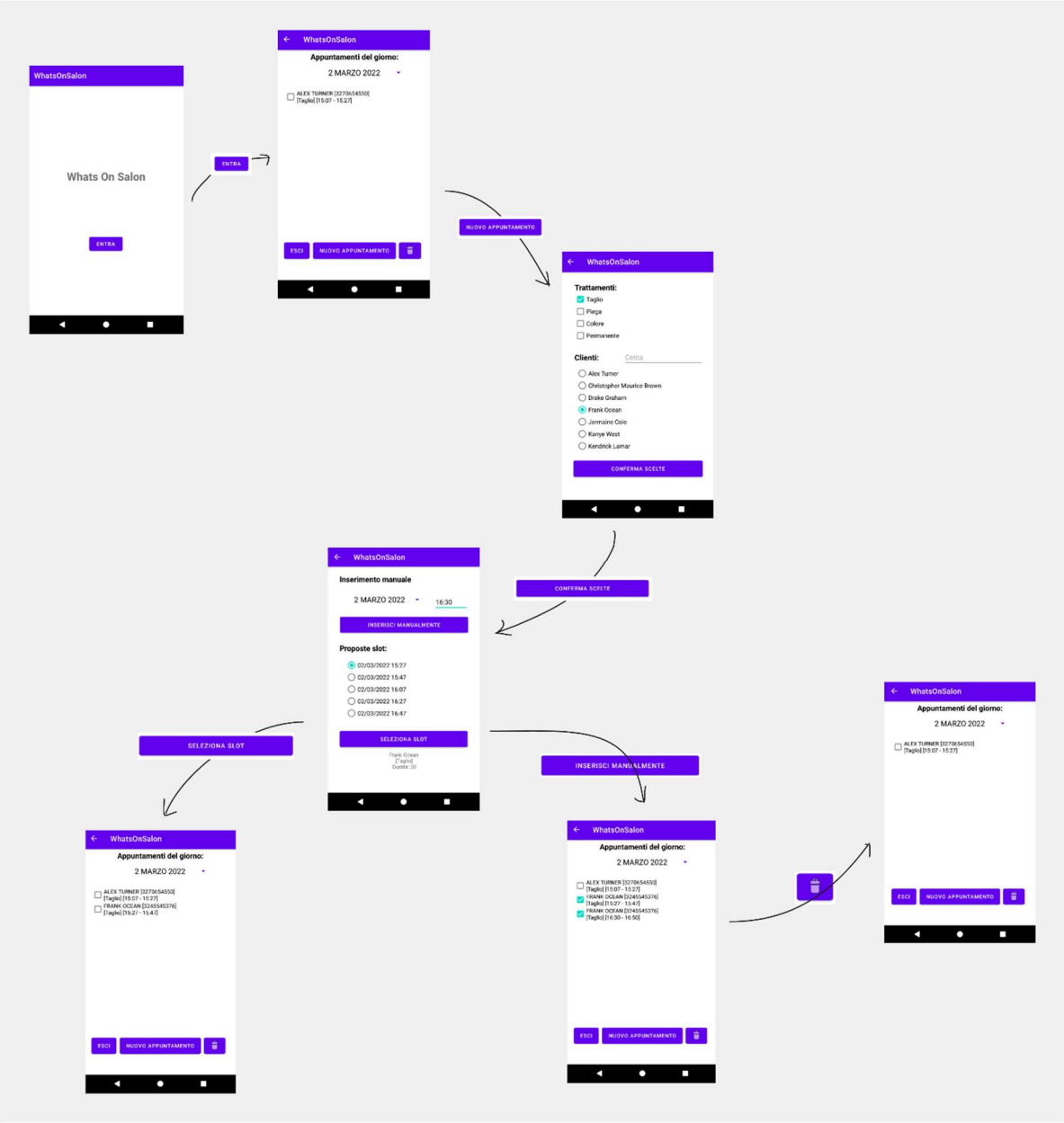
La complessità dell'algoritmo è Ω(nSlot), caso in cui tutte le proposte vengono trovate al primo tentativo e O(nSlot * nAppuntamentiFuturi), caso in cui viene raggiunta una prossimaProposta successiva alla fine dell'ultimo appuntamento futuro solo nel corso dell'ultima iterazione del while.

Fortunatamente, tutti gli onerosi calcoli degli orari di fine appuntamento presenti nello pseudocodice sono stati semplificati dall'introduzione nel data model degli appuntamenti di un campo dataFine.

Si veda di seguito il flow chart dell'algoritmo.


FUNC propostaSlotAppuntamento(appuntamenti, durata, nSlot)
ordinaCrescente(appuntamenti, dataOra)
DateTime[nSlot] proposta = 0
DateTime prossimaProposta = adesso()
int slotProposti = 0
SlotProposti < nSlot
F
T
DateTime fineUltimoAppuntamento = appuntamenti[appuntamenti.length - 1].dataOra
Trattamento t; t in appuntamenti[appuntamenti.length - 1].trattamenti; next t
F
T
fineUltimoAppuntamento = fineUltimoAppuntamento + t.durata
F
proposta[slotProposti] < fineUltimoAppuntamento
T
Appuntamento a; a in appuntamenti; next a
F
T
F
a.dataOra >= proposta[nSlot] + durata && proposta[nSlot] <= ORARIO_CHIUSURA
T
GOTO break
proposta[nSlot] = a.dataOra
Trattamento t; t in trattamenti; next t
F
T
proposta[nSlot] = proposta[nSlot] + t.durata
break
prossimaProposta = proposta[nSlot] + durata
slotProposti = slotProposti + 1
RET proposta

3.4 RISULTATI IMPLEMENTAZIONE



3.5 ANALISI STATICA

Per l'analisi statica del codice è stato utilizzato un plug-in disponibile in Android Studio chiamato MetricsReloaded che fornisce le metriche del codice sorgente in esame.

Il plugin fornisce delle misure di complessità del codice. In particolare:

- **CogC:** Cognivite Complexity

  Misura la complessità in termini di comprensione del codice. Questo valore aumenta all'aumentare del numero di strutture controllate e strutture annidate.

- **ev(G)**: Essential Cyclomatic Complexity
  Misura quanto sia mal strutturato il flusso di controllo di un metodo.
- **iv(G)**: Design Complexity
  Calcola la complessità di progettazione di un metodo. La complessità della progettazione è correlata al modo in cui un flusso di controllo dei metodi è interconnesso con le chiamate ad altri metodi.
- **v(G)**: Cyclomatic Complexity
  È una misura del numero di percorsi di esecuzione distinti attraverso ciascun metodo. Questo può anche essere considerato come il numero minimo di prove necessarie per esercitare completamente il flusso di controllo di un metodo. In pratica, questo è 1 + il numero di if, while, for, do, switch case, catch, espressioni condizionali, and e or nel metodo.

L'analisi con MetricsReloaded ha dato i seguenti risultati, ordinati in base al valore decrescente di v(G).

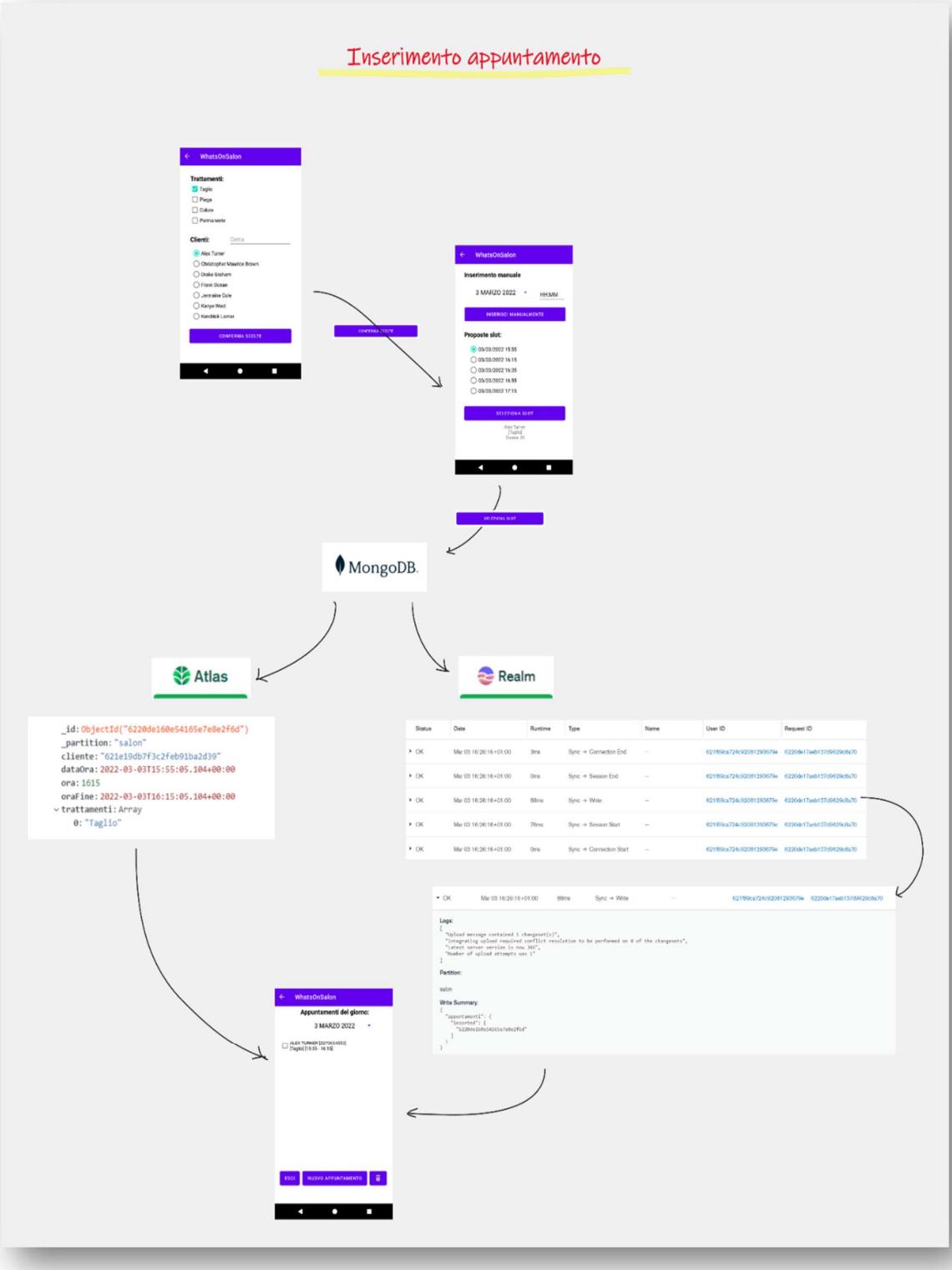
method	CogC	ev(G)	iv(G)	v(G)
com.example.whatsonsalon.SelezionaSlotAppuntamento.proponiSlot(SyncConfiguration)	40	1	15	17
com.example.whatsonsalon.AggiungiAppActivity.onCreate(Bundle)	58	1	14	14
com.example.whatsonsalon.MainPage.getMonthFormat(int)	12	13	1	13
com.example.whatsonsalon.SelezionaSlotAppuntamento.getMonthFormat(int)	12	13	1	13
com.example.whatsonsalon.SelezionaSlotAppuntamento.onCreate(Bundle)	48	1	12	13
com.example.whatsonsalon.MainPage.popolaTabella(SyncConfiguration, boolean)	21	1	11	12
com.example.whatsonsalon.AggiungiAppActivity.popola(SyncConfiguration)	5	1	3	6
com.example.whatsonsalon.MainPage.onCreate(Bundle)	17	1	5	6
com.example.whatsonsalon.AggiungiAppActivity.trovaIndice(RadioButton[], int)	7	2	2	5
com.example.whatsonsalon.SelezionaSlotAppuntamento.settaInfo(SyncConfiguration)	6	3	5	5
com.example.whatsonsalon.SelezionaSlotAppuntamento.trovaIndice(RadioButton[], int)	7	2	2	5
com.example.whatsonsalon.MainPage.mostraOrario(long)	3	1	3	3
com.example.whatsonsalon.MainPage.trovaCliente(String, RealmResults<clienti>)	3	3	3	3
com.example.whatsonsalon.SelezionaSlotAppuntamento.riempiScrollView()	2	1	2	3
com.example.whatsonsalon.MainPage.onDestroy()	3	1	2	2
com.example.whatsonsalon.MainActivity.onCreate(Bundle)	0	1	1	1
com.example.whatsonsalon.MainPage.esci(View)	0	1	1	1
com.example.whatsonsalon.MainPage.getTodaysDate()	0	1	1	1
com.example.whatsonsalon.MainPage.initDatePicker()	0	1	1	1
com.example.whatsonsalon.MainPage.makeDateString(int, int, int)	0	1	1	1
com.example.whatsonsalon.MainPage.openDatePicker(View)	0	1	1	1
com.example.whatsonsalon.SelezionaSlotAppuntamento.getTodaysDate()	0	1	1	1
com.example.whatsonsalon.SelezionaSlotAppuntamento.initDatePicker()	0	1	1	1
com.example.whatsonsalon.SelezionaSlotAppuntamento.makeDateString(int, int, int)	0	1	1	1
com.example.whatsonsalon.SelezionaSlotAppuntamento.openDatePicker2(View)	0	1	1	1
com.example.whatsonsalon.appuntamenti.getCliente()	0	1	1	1
com.example.whatsonsalon.appuntamenti.getDataOra()	0	1	1	1
com.example.whatsonsalon.appuntamenti.getOra()	0	1	1	1
com.example.whatsonsalon.appuntamenti.getOraFine()	0	1	1	1

com.example.whatsonsalon.SelezionaSlotAppuntamento.proponiSlot(syncConfiguration)
Per questo metodo si hanno alti valori per Cognitive Complexity, Design Complexity e Cyclomatic Complexity, giustificati dalla presenza di strutture di controllo annidate e chiamate ad altri metodi.
Per interpretare al meglio il valore di Cyclomatic Complexity è necessario considerare il valore di Essential Cyclomatic Complexity. Il primo parametro può essere interpretato come difficoltà di testing del metodo in quanto rappresenta il numero di percorsi di esecuzione distinti, definiti dalla presenza di punti di decisione (if, while, for, do, switch case, catch, espressioni condizionali, and e or). Per il metodo in esame ci sono 17 possibili percorsi da testare. In genere, 1-4 è una complessità bassa, 5-7 indica una complessità moderata, 8-10 è una complessità elevata e 11+ è una complessità molto elevata.
L'elevata complessità ciclomatica viene però compensata dal valore unitario della complessità essenziale. Questo parametro, che indica quanto sia mal strutturato il flusso di controllo di un metodo, indica che è relativamente semplice scomporre il metodo in sotto-metodi più semplici da testare.
Le considerazioni fatte per questo metodo si riflettono anche per gli altri metodi con stessi risultati.

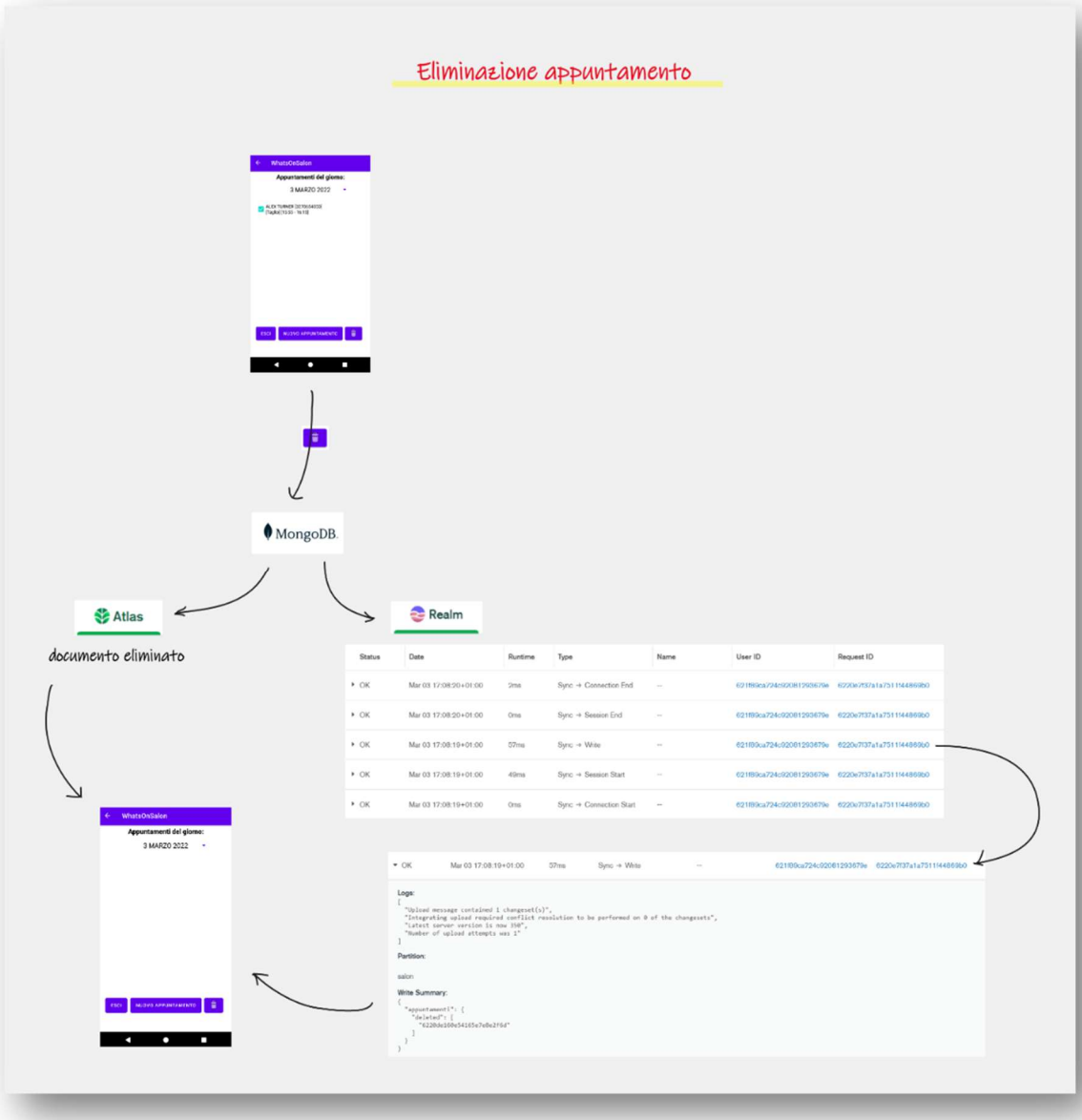
3.6 ANALISI DINAMICA

Per effettuare un'analisi dinamica del progetto si è preferito verificare il corretto funzionamento delle chiamate al database. Per facilitare questa fase ci si è affidati alla console di logs offerto dalla piattaforma Realm e alla verifica manuale degli inserimenti/cancellazioni dei documenti dal data lake Atlas.
A tal proposito si è simulato un inserimento di un appuntamento e alla sua successiva eliminazione dal database.

Inserimento appuntamento:



Cancellazione appuntamento:



ITERAZIONE 3

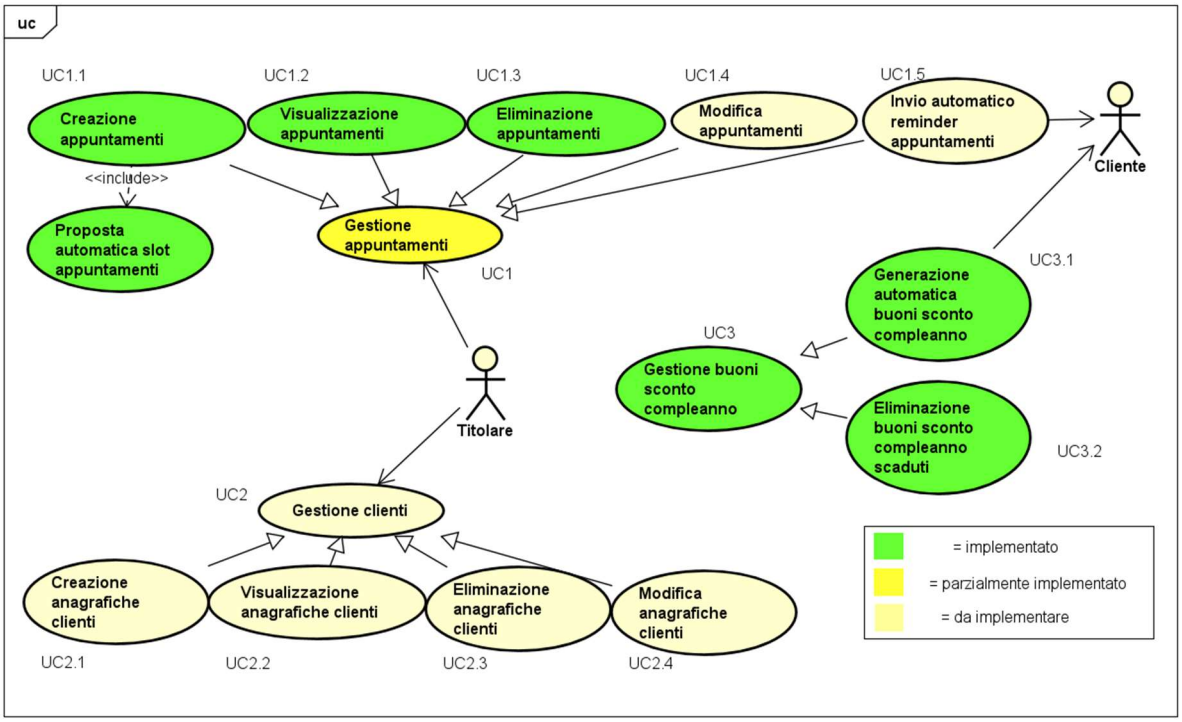
4.1 CASI D'USO SELEZIONATI

Nella terza iterazione si è deciso di implementare i seguenti casi d'uso:

- UC3: Gestione buoni sconto compleanno
    - UC3.1: Generazione automatica buoni sconto compleanno;
    - UC3.2: Eliminazione automatica buoni sconto compleanno scaduti;

Oltre all'implementazione dei suddetti UC, sono state apportate migliorie di entità minima ma di enorme utilità all'interfaccia grafica di WhatsOnSalon.

Dal diagramma dei casi d'uso sottostante è possibile analizzare l'avanzamento del progetto facendo riferimento alla legenda in basso a destra.



UC3.1: Generazione automatica buoni sconto compleanno

Descrizione: in occasione dei compleanni dei clienti, il sistema deve generare automaticamente buoni sconto utilizzabili entro due mesi dalla data del compleanno e inviarli.

Oltre alla scadenza, deve essere associato ad ogni buono anche un codice univoco.

Attori coinvolti: trigger temporale.

Trigger: arrivo della mezzanotte.

Postcondizione: il database dei buoni sconto conterrà un nuovo buono sconto della valenza di due mesi per ogni cliente che compie gli anni nella giornata odierna.

Procedimento: 1. allo scatto della mezzanotte, l'algoritmo di gestione buoni sconto viene lanciato;

2. l'algoritmo recupera gli _id dei clienti che compiono gli anni quel giorno;

3. per ogni _id recuperato, viene generato un buono sconto di codice univoco della valenza di due mesi (a partire dal giorno stesso);

4. per ogni buono sconto generato, viene inviato un messaggio di auguri e di notifica generazione buono al numero di telefono del cliente per cui è stato emesso;

UC3.2: Eliminazione automatica buoni sconto compleanno scaduti

Descrizione: il sistema deve eliminare automaticamente i buoni sconto scaduti non appena possibile.

Attori coinvolti: trigger temporale.

Trigger: arrivo della mezzanotte.

Postcondizione: il database dei buoni sconto conterrà solo buoni non ancora scaduti.

Procedimento: 1. allo scatto della mezzanotte, l'algoritmo di gestione buoni sconto viene lanciato;

2. l'algoritmo confronta la data odierna con la data di scadenza di ogni buono sconto nell'apposito database, andando a eliminare i buoni che risultano scaduti;

Migliorie all'interfaccia grafica di WhatsOnSalon

- possibilità di ricerca cliente per nome e/o cognome durante la creazione di un appuntamento;
- predisposizione per l'aggiunta di nuovi clienti al database durante la creazione di un appuntamento – pulsante che sarà programmato per questo scopo nelle iterazioni

successive;

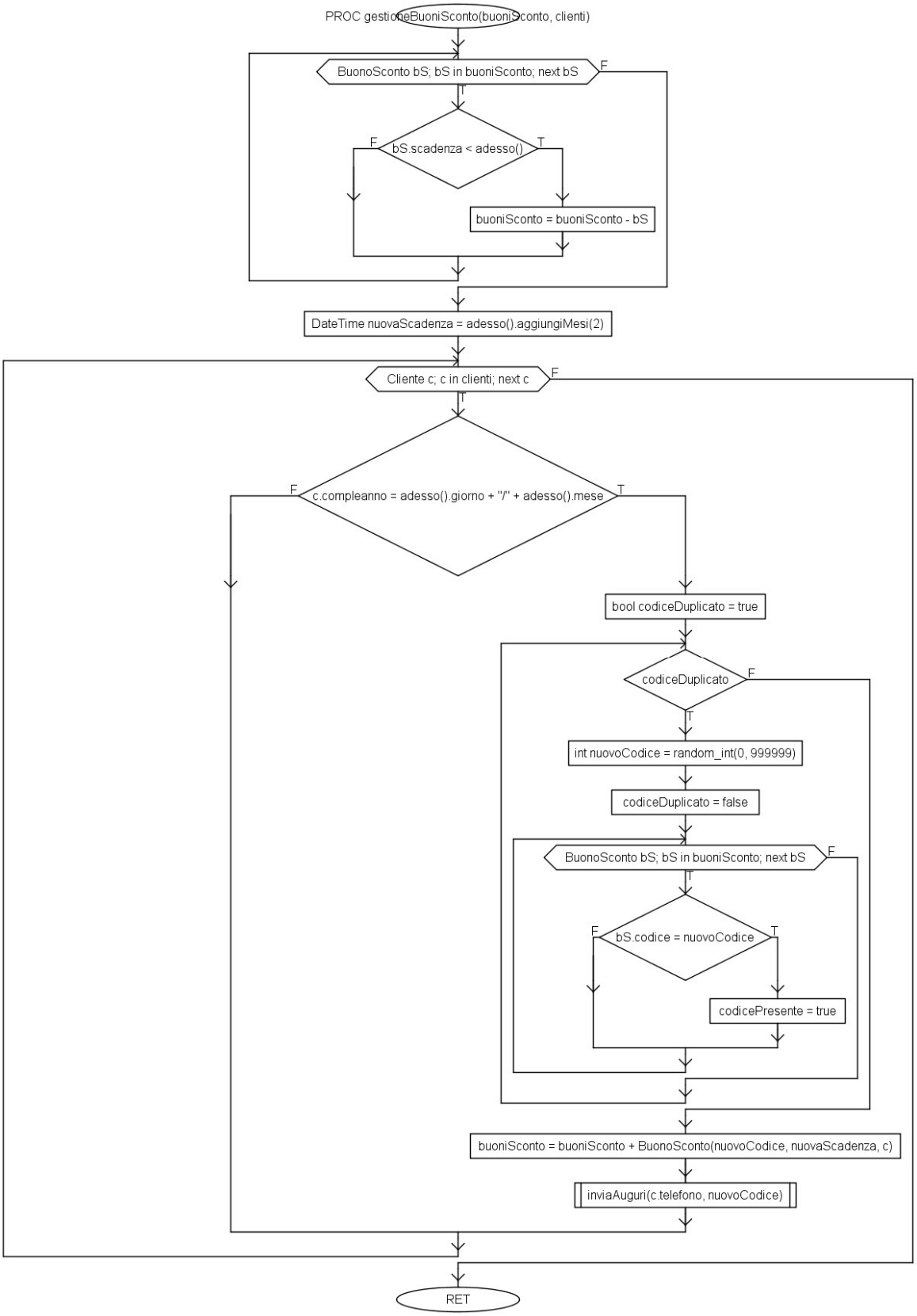
- aggiunta numero di telefono del cliente tra le informazioni mostrate nel riepilogo degli appuntamenti in programma;
- piccole variazioni nella presentazione grafica dell'app;

4.2 PSEUDOCODICE

UC3: Gestione buoni sconto compleanno

```
in: elenco buoni sconto esistenti, elenco clienti esistenti
out:
gestioneBuoniSconto(BuonoSconto[] buoniSconto, Cliente[] clienti) -> void
    rimozione buoni sconto scaduti
    for BuonoSconto bS in buoniSconto
        if bS.scadenza < adesso()
            buoniSconto <- buoniSconto - bS


    generazione buoni sconto
    DateTime nuovaScadenza <- adesso().aggiungiMesi(2)
    for Cliente c in clienti
        c.compleanno è salvata come stringa formato dd/mm
        if c.compleanno = adesso().giorno + "/" + adesso().mese
            bool codiceDuplicato <- true
            while codiceDuplicato <-- garanzia univocità codice del buono in generazione
                int nuovoCodice <- random_int(0, 999999) <-- generazione intero tra 0 e 999999 compresi
                codiceDuplicato <- false
                for BuonoSconto bS in buoniSconto
                    if bS.codice = nuovoCodice
                        codicePresente <- true

            supponiamo esista un costruttore BuonoSconto(int codice, string scadenza, Cliente titolare)
            buoniSconto <- buoniSconto + BuonoSconto(nuovoCodice, nuovaScadenza, c)
            supponiamo esista un metodo per l'invio di auguri + notifica generazione buono
            inviaAuguri(c.telefono, nuovoCodice)
```

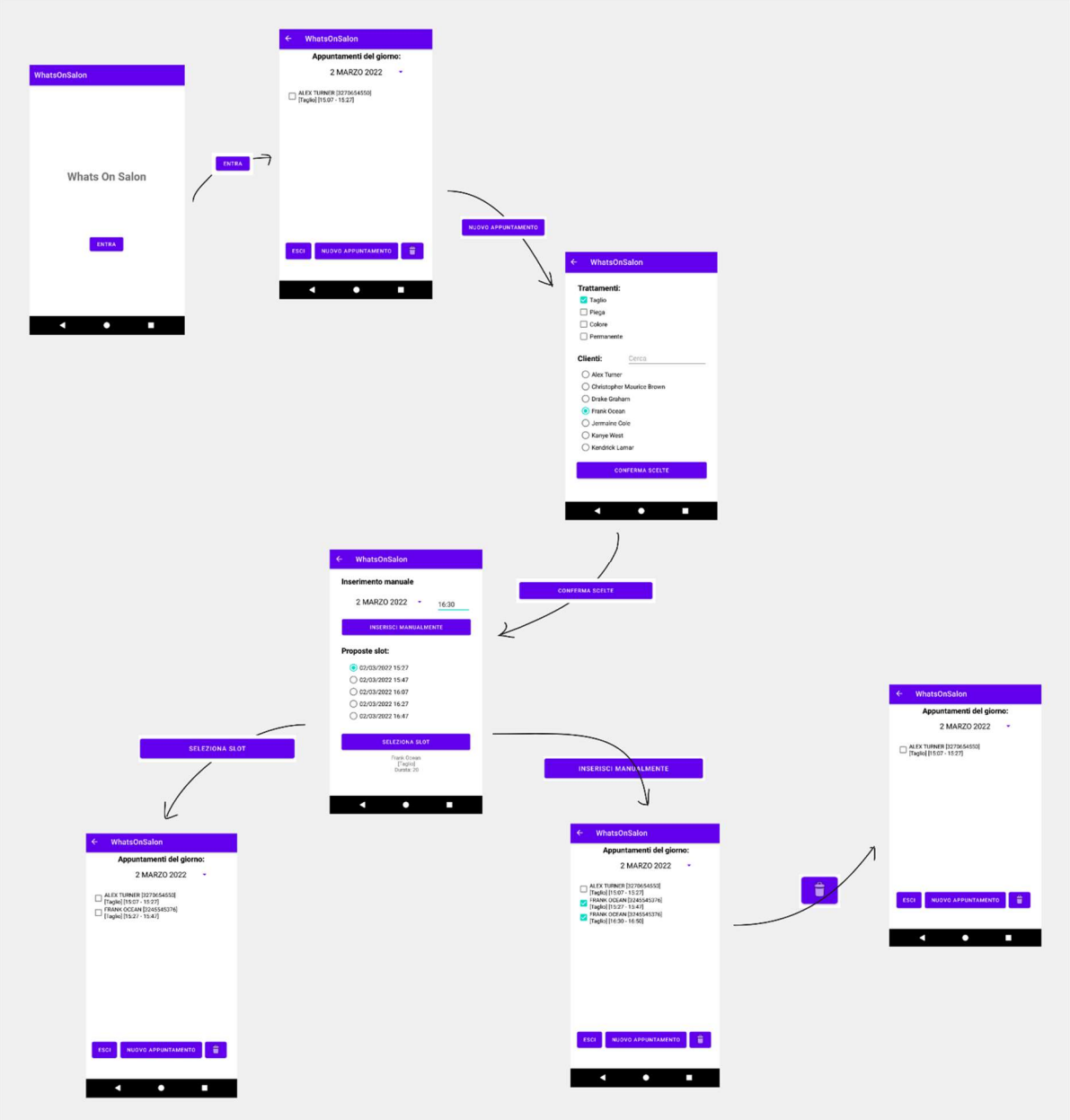
La complessità dell'algoritmo è Ω(nBuoniSconto), caso in cui non ci sono né buoni da cancellare né buoni da generare e si esegue solo il primo for, e O(nBuoniSconto + nClienti), caso in cui - indipendentemente dal numero di buoni sconto da eliminare – tutti i clienti compiono gli anni e bisogna generare un buono per ognuno, dando così luogo a nClienti iterazioni del secondo for.
Nell'implementazione vera e propria i for che scandagliano i database di buoni sconto e clienti sono sostituiti da API call tramite il Node.js MongoDB driver, che hanno complessità inferiore e abbassano dunque la complessità teorica dell'algoritmo.
Si veda di seguito il flow chart dell'algoritmo.


PROC gestioneBuoniSconto(buoniSconto, clienti)
BuonoSconto bS; bS in buoniSconto; next bS
F
T
F
bS.scadenza < adesso()
T
buoniSconto = buoniSconto - bS
DateTime nuovaScadenza = adesso().aggiungiMesi(2)
Cliente c; c in clienti; next c
F
T
F
c.compleanno = adesso().giorno + "/" + adesso().mese
T
bool codiceDuplicato = true
codiceDuplicato
F
T
int nuovoCodice = random_int(0, 999999)
codiceDuplicato = false
BuonoSconto bS; bS in buoniSconto; next bS
F
T
F
bS.codice = nuovoCodice
T
codicePresente = true
buoniSconto = buoniSconto + BuonoSconto(nuovoCodice, nuovaScadenza, c)
inviaAuguri(c.telefono, nuovoCodice)
RET

4.3 RISULTATI IMPLEMENTAZIONE

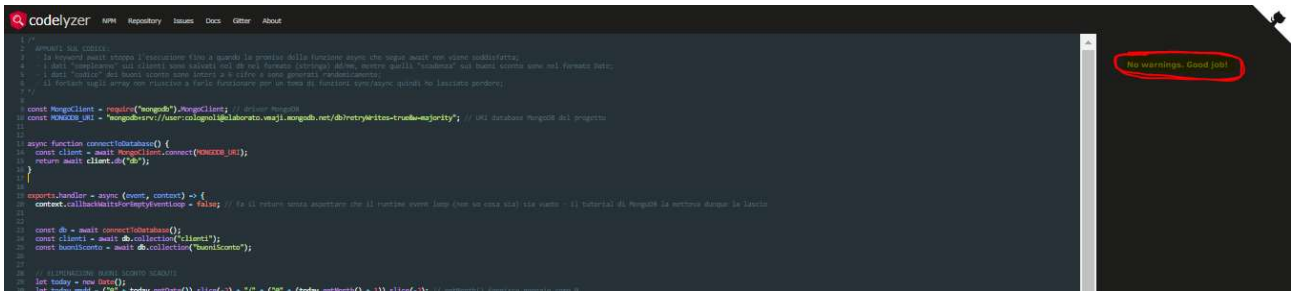
Tutto ciò che riguarda l'UC3 non ha alcun impatto sull'interfaccia grafica di WhatsOnSalon né introduce alcun nuovo ambiente accessibile da parte del titolare: tutto è gestito automaticamente e mostrare i risultati è più di interesse della fase di testing che della descrizione della user experience.

Per quanto riguarda le migliorie all'UI di WhatsOnSalon, che invece non necessitano di testing dato che non cambiano il comportamento dell'app, ecco i risultati (non molto diversi da quelli osservabili nell'iterazione 2).



4.4 TESTING

Per testare il corretto funzionamento di gestioneBuoniSconto, si è partiti dall'analisi statica della sua implementazione Node.js tramite l'apposito tool codelyzer, che non ha evidenziato warning o errori nel codice.



Con la sicurezza che il codice rispettasse buoni standard, si è potuto procedere con l'analisi dinamica dello stesso.

Innanzitutto, è stato creato un ambiente di test, così strutturato (data test: 07/03/2022):

- database buoni sconto avente sia buoni sconto scaduti che buoni sconto validi, con copertura casi limite:
    - scadenza = data test (dovrebbe essere eliminato);
    - scadenza = data test + 1 giorno (non dovrebbe essere eliminato);

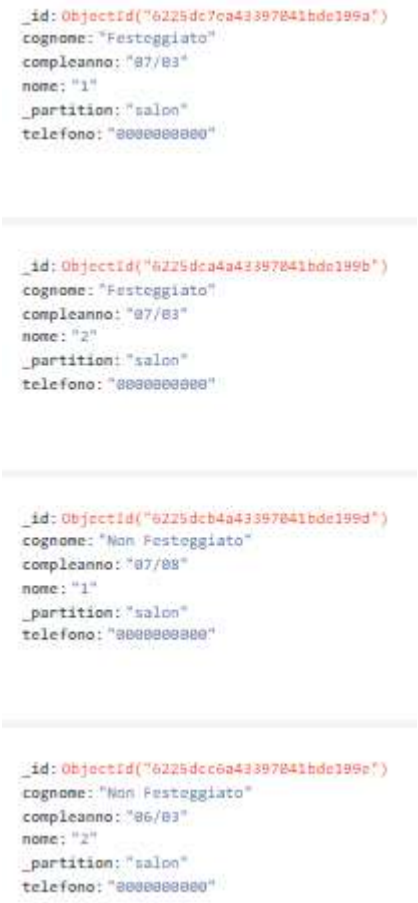
```
_id: ObjectId("6225dae5a43397041bde1999")
codice: "111110"
scadenza: 2022-08-06T00:00:00.000+00:00
titolare: "questo non è scaduto"
```

```
_id: ObjectId("6225df37a43397041bde19a1")
codice: "111111"
scadenza: 2022-03-08T00:00:00.000+00:00
titolare: "questo non è scaduto - caso limite"
```

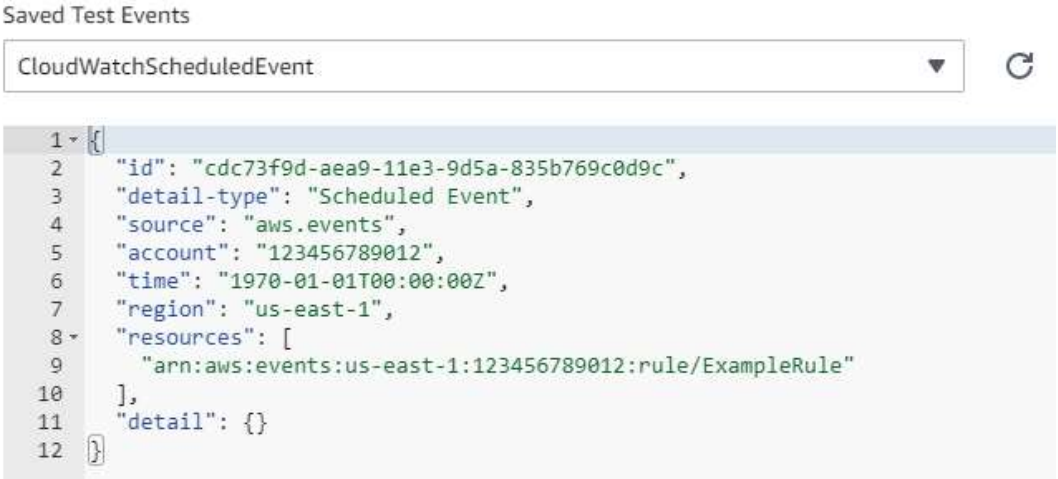
```
_id: ObjectId("6225df56a43397041bde19a3")
codice: "000001"
scadenza: 2022-03-07T00:00:00.000+00:00
titolare: "questo è scaduto - caso limite"
```

```
_id: ObjectId("6225df7ba43397041bde19a5")
codice: "000000"
scadenza: 2021-08-07T00:00:00.000+00:00
titolare: "questo è scaduto"
```

- database clienti avente sia clienti che compiono gli anni nella data di testing che clienti che nati in altre date (sempre coprendo i due casi limite sopra enunciati);



- trigger temporale che simuli lo scatto della mezzanotte;



Idealmente, i risultati avrebbero dovuto essere i seguenti:

- database buoni sconto popolato dai buoni sconto validi tra quelli precedentemente esistenti e da quelli generati per ogni cliente festeggiato nella data di testing;
- database clienti invariato;

- trigger temporale che innesca esattamente un’esecuzione del codice;

Di fatto, così è stato:


```
_id: ObjectId("6225dae5a43397041bde1999")
codice: "111110"
scadenza: 2022-08-06T00:00:00.000+00:00
titolare: "questo non è scaduto"
```

```
_id: ObjectId("6225df37a43397041bde19a1")
codice: "111111"
scadenza: 2022-03-08T00:00:00.000+00:00
titolare: "questo non è scaduto - caso limite"
```

```
_id: ObjectId("6225e0e0d1451e32eaaf995c")
codice: 551380
scadenza: 2022-05-07T00:00:00.000+00:00
titolare: ObjectId("6225dc7ea43397041bde199a")
```

```
_id: ObjectId("6225e0e1d1451e32eaaf995d")
codice: 158277
scadenza: 2022-05-07T00:00:00.000+00:00
titolare: ObjectId("6225dca4a43397041bde199b")
```

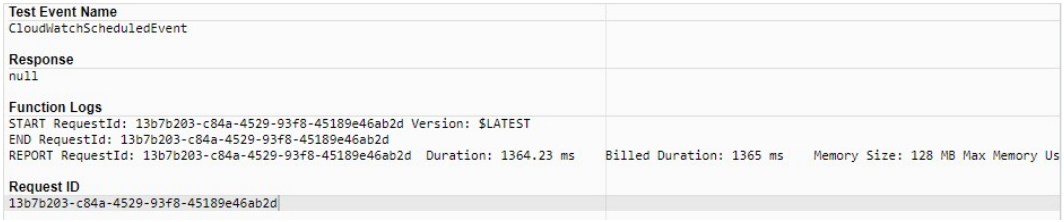
-
```
_id: ObjectId("6225dc7ca43397041bde199a")
cognome: "Festeggiato"
compleanno: "07/03"
nome: "1"
_partition: "salon"
telefono: "0000000000"
```

```
_id: ObjectId("6225dca4a43397041bde199b")
cognome: "Festeggiato"
compleanno: "07/03"
nome: "2"
_partition: "salon"
telefono: "0000000000"
```

```
_id: ObjectId("6225dcb4a43397041bde199d")
cognome: "Non Festeggiato"
compleanno: "07/08"
nome: "1"
_partition: "salon"
telefono: "0000000000"
```

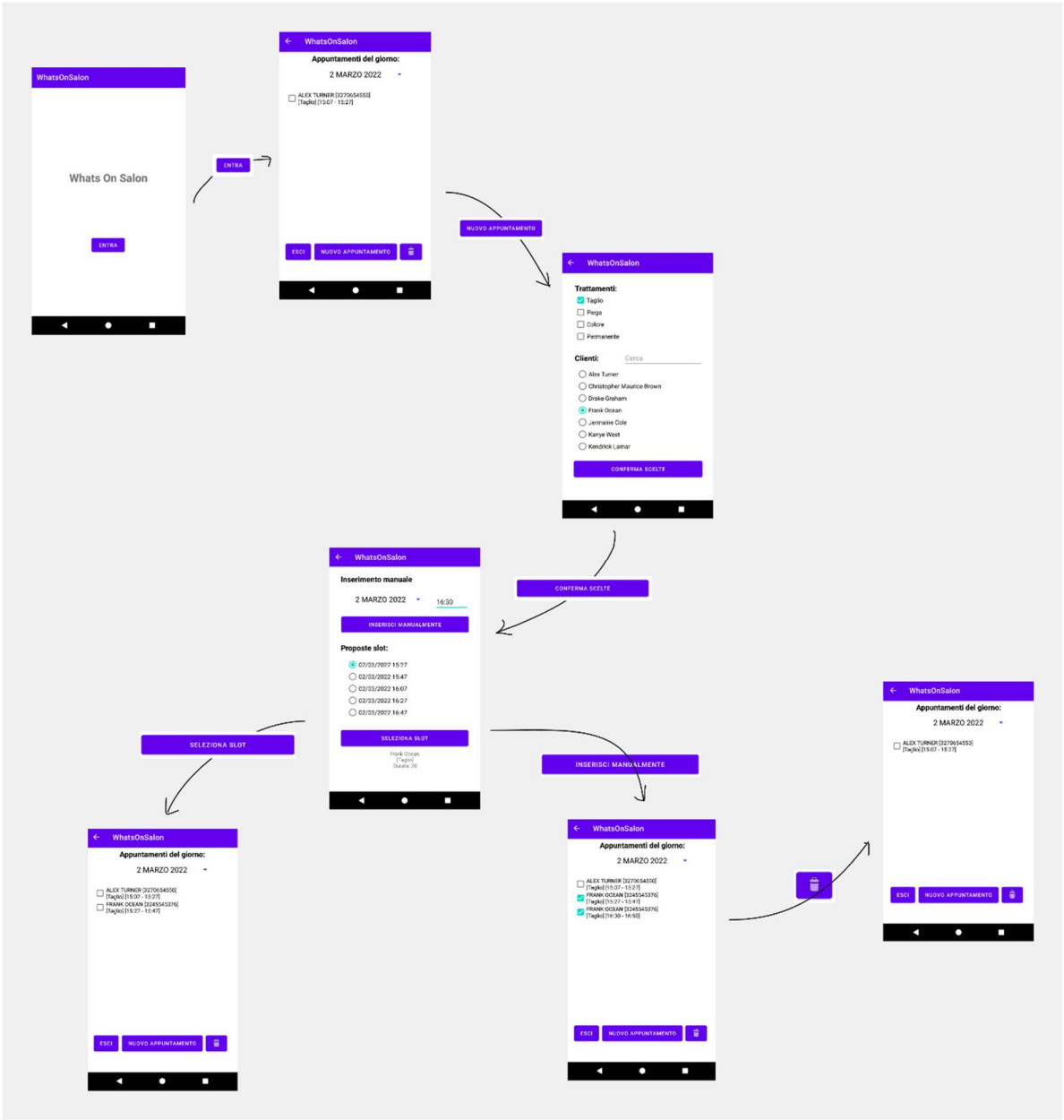
```
_id: ObjectId("6225dcc6a43397041bde199e")
cognome: "Non Festeggiato"
compleanno: "06/03"
nome: "2"
_partition: "salon"
telefono: "0000000000"
```

- (tempo di esecuzione: 1364.23ms)



Si può dunque concludere che lo sforzo iterativo è andato a buon fine e ci si può organizzare per intraprendere l'iterazione successiva.

MANUALE D’USO


WhatsOnSalon
Whats On Salon
ENTRA
ENTRA
WhatsOnSalon
Appuntamenti del giorno:
2 MARZO 2022
ESCI
NUOVO APPUNTAMENTO
NUOVO APPUNTAMENTO
WhatsOnSalon
Trattamenti:
Taglio
Piega
Colore
Permanente
Clienti:
Cerca
Alex Turner
Christopher Maurice Brown
Drake Graham
Frank Ocean
Jermaine Cole
Kanye West
Kendrick Lamar
CONFERMA SCELTE
CONFERMA SCELTE
WhatsOnSalon
Inserimento manuale
2 MARZO 2022
16:30
INSERISCI MANUALMENTE
Proposte slot:
02/03/2022 15:27
02/03/2022 15:47
02/03/2022 16:07
02/03/2022 16:27
02/03/2022 16:47
SELEZIONA SLOT
SELEZIONA SLOT
INSERISCI MANUALMENTE
WhatsOnSalon
Appuntamenti del giorno:
2 MARZO 2022
ESCI
NUOVO APPUNTAMENTO
WhatsOnSalon
Appuntamenti del giorno:
2 MARZO 2022
ESCI
NUOVO APPUNTAMENTO
WhatsOnSalon
Appuntamenti del giorno:
2 MARZO 2022
ESCI
NUOVO APPUNTAMENTO

BIBLIOGRAFIA

Nella realizzazione di questo progetto è stato fatto riferimento ai seguenti siti web:

https://docs.mongodb.com/realm/sdk/

https://docs.mongodb.com/realm/sdk/android/

https://docs.mongodb.com/drivers/node/current/

https://www.twilio.com/docs/sms/send-messages

https://www.twilio.com/docs/sms/api

http://codelyzer.com/