ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

PROGETTAZIONE E SVILUPPO DI UN SISTEMA SOFTWARE PER IL TICKETING

Relazione finale in
PROGRAMMAZIONE DI SISTEMI MOBILE

Relatore
Dott. MIRKO RAVAIOLI

Presentata da
ANNA GIULIA LEONI

Seconda Sessione di Laurea
Anno Accademico 2014 – 2015

PAROLE CHIAVE

Ticket

Android

App

Ai miei genitori

Indice

Introduzione **ix**

1 Sguardo generale sul sistema **1**
1.1 Il mondo mobile 1
1.2 Concetti fondamentali per la comprensione del sistema 1

2 Analisi del sistema **5**
2.1 Funzionalità principali 5
2.1.1 Apertura di un ticket 5
2.1.2 Assegnazione di un ticket 6
2.1.3 Gestione di un ticket 7
2.1.4 Visualizzazione di un progetto/ordine/attività 7
2.1.5 Aggiunta di un progetto/ordine/attività/utente 7
2.1.6 Gestione del profilo 7
2.2 Diagramma dei casi d'uso 8
2.3 Sistemi simili già esistenti 8

3 Progettazione **11**
3.1 Progettazione concettuale 11
3.1.1 Analisi di entità ed associazioni 11
3.1.2 Schema concettuale 15
3.2 Progettazione logica 16
3.2.1 Eliminazione della gerarchia 16
3.2.2 Traduzione delle associazioni 18
3.2.3 Schema logico 20
3.3 Dati derivati 20
3.4 Web Server 21
3.5 Mockup 23

4 Implementazione **29**
4.1 Server 29
4.1.1 Strumenti utilizzati 29

4.1.2 Pattern di comunicazione tra Client e Server 30
4.1.3 Esempi di risposta servizi lato server 31
4.2 Client 35
4.2.1 Sviluppo di un'app nativa 35
4.2.2 Android come sistema operativo mobile 36
4.2.3 Android Studio come ambiente di sviluppo 38
4.2.4 Modalità di utilizzo 38
4.2.5 Suddivisione in moduli 39
4.2.6 Esempio di richiesta dati al server 40
4.2.7 Scelte implementative 43
4.2.8 Screenshot schermate principali 47

5 Testing **51**
5.1 Latenza 51

Conclusioni **53**

Ringraziamenti **55**

Bibliografia **59**

Introduzione

La presente tesi ha come obiettivo quello di descrivere la progettazione e lo sviluppo di un applicazione per dispositivi mobile la cui funzionalità principale è quella di gestire segnalazioni in merito a problematiche riscontrate dall'utente che possono riguardare diversi ambiti (segnalazioni post-vendita in merito a progetti informatici, segnalazioni in merito a disguidi su ordini). Tale applicazione rappresenta una versione base che potrà poi essere modificata e specializzata nella gestione di segnalazioni relative ad un solo ambito, a seconda delle necessità del cliente.

Lo scopo del progetto di tesi non è relativo al solo sviluppo del software ma è volto, nella sua versione completa, ad uniformare il canale di comunicazione tra azienda e cliente e a fornire all'azienda che ne ha fatto richiesta un sistema efficiente per la gestione di queste segnalazioni.

Al momento, la comunicazione tra le due parti avviene tramite più tradizionali metodi di comunicazione come il telefono o la posta elettronica. Questi metodi tuttavia non permettono di tenere traccia in maniera ordinata della segnalazione perché il cliente potrebbe, ad esempio, segnalare un problema tramite posta elettronica, dimenticarsi di inserire alcuni dettagli, e decidere di rimediare con una telefonata. In questo modo l'impiegato che controlla la posta e prende in carico la segnalazione potrebbe non essere lo stesso che ha risposto al telefono ed entrambi sarebbero in possesso di informazioni incomplete.

Un sistema informatico volto alla registrazione di queste segnalazioni invece, elimina queste problematiche e ne facilita e velocizza la gestione, portando dei benefici ad entrambi gli utilizzatori, impiegati dell'azienda e clienti.

Il cliente, tramite questa applicazione, potrà segnalare nuovi problemi riscontrati, essere sempre aggiornato sullo stato delle sue segnalazioni e fornire un feedback all'azienda in merito al lavoro svolto.

L'impiegato dell'azienda, invece, sarà in grado di seguire il problema del cliente, fornendogli un supporto immediato, aumentando la customer satisfaction.

La tesi è suddivisa in 4 capitoli:

Il primo capitolo presenta uno sguardo generale sul mondo mobile ed introduce i concetti fondamentali per la comprensione del sistema.

Il secondo capitolo si occupa dell'analisi delle funzionalità richieste dal sistema e prende in esame sistemi simili già esistenti.

Il terzo capitolo si occupa della progettazione del software, dallo schema concettuale a quello logico, evidenziando la funzione di ogni entità e le relazioni tra queste.

Il quarto capitolo si occupa dell'implementazione, dalla scelta del sistema operativo su cui realizzare l'applicazione alla descrizione delle principali scelte implementative client e server.

Seguono conclusioni, sviluppi futuri e ringraziamenti.

Capitolo 1

Sguardo generale sul sistema

1.1 Il mondo mobile

L'importanza assunta dal mondo mobile negli ultimi anni ha rivoluzionato la vita quotidiana di milioni di persone in tutto il mondo e la loro esperienza in rete. Sempre più utenti stanno migrando dal mondo desktop a quello mobile e per le aziende uno degli strumenti più efficaci per raggiungere i propri clienti sono diventate le app.

Gli smartphone sono ad oggi i dispositivi mobile più diffusi. Permettono di compiere, dal palmo della propria mano, le più svariate funzioni, oltre a quelle proprie di telefono cellulare: ricerche e acquisti online, visualizzazione e download di contenuti multimediali, utilizzo di Social Network...

A confermare il ruolo che gli smartphone hanno raggiunto sono le parole di Amit Singhal, vicepresidente senior della divisione Search di Google. Durante la conferenza annuale "Code Mobile" ha dichiarato che, per la prima volta, sulle oltre 100 miliardi di ricerche su Google effettuate in tutto il pianeta, oltre il 50% sono state effettuate da smartphone [1].

Le considerazioni appena riportate hanno indirizzato lo sviluppo del sistema appositamente per dispositivi mobile.

1.2 Concetti fondamentali per la comprensione del sistema

Alla luce di quanto verrà esposto in seguito in merito alla progettazione del sistema, vengono definiti in questa sezione alcuni concetti fondamentali dai quali non si può prescindere per la comprensione del sistema stesso.

Da questo momento in poi, le richieste di assistenza che il cliente, attraverso un sistema informatico, inoltra all'azienda per la risoluzione di una problematica, verranno identificate nel **ticket**. Il ticket rappresenterà il fulcro del sistema che si andrà a progettare.

I dati relativi ai ticket e agli utilizzatori del sistema verranno immagazzinati in un una **base di dati** o **database**. Un database è una collezione di dati tra loro collegati. I database sono strutturati secondo un modello logico, scelto dal progettista a seconda delle esigenze. Per interfacciarsi con i database, gli utenti utilizzano i DBMS e linguaggi di interrogazione.

Un **DBMS** (DataBase Management System) è un sistema software che gestisce grandi quantità di dati, persistenti e condivisi, garantendone l'integrità e la sicurezza mediante l'esecuzione coordinata delle richieste, la protezione da malfunzionamenti e la realizzazione di una politica degli accessi [2].

I **linguaggi di interrogazione** sono linguaggi che permettono l'estrazione di informazioni dal database, attraverso i DBMS, sulla base delle richieste effettuate dall'utente.

Il database che verrà progettato sarà funzionale all'applicazione. Questa, infatti, potrà ottenere tutti i dati necessari al suo funzionamento dal database tramite l'ausilio di un Server Web. Un **Server Web** è un software che, utilizzando il modello client/server e il protocollo HTTP, è in grado di gestire le richieste di un client (in questo caso l'applicazione) e formulare una risposta in modo dinamico.

La sequenza di operazioni che avviene ogni volta che l'applicazione richiede dati al database è riportata nella figura sottostante.

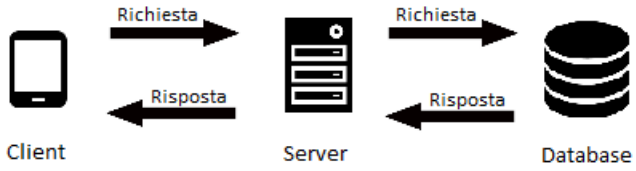


Figura 1.1: Schema interfacciamento applicazione e database

Nel corso di questa tesi verrà inoltre utilizzato il formalismo **UML** (Unified Modeling Language), un linguaggio di modellazione della realtà orientato agli oggetti che fornisce i costrutti per modellare le diverse fasi dello sviluppo software (analisi dei requisiti e progettazione del sistema).

E' basato su diagrammi (rappresentazioni grafiche della realtà da modellare) e può essere utilizzato indipendentemente dall'ambito del progetto, dal processo di sviluppo e dal linguaggio di programmazione.

Concluso questo elenco di strumenti che verranno utilizzati per la progettazione e lo sviluppo dell'applicazione si può procedere con una analisi più approfondita delle sue funzionalità.

Capitolo 2

Analisi del sistema

In questo capitolo vengono descritte le funzionalità principali del sistema richiesto, individuati gli attori coinvolti e la loro interazione con il sistema stesso.

Prima di passare alla progettazione vengono presi in esame sistemi simili ed effettuato un breve confronto sulla base delle loro funzionalità.

2.1 Funzionalità principali

2.1.1 Apertura di un ticket

L'apertura di un nuovo ticket consiste nell'inserimento di alcuni dati relativi al problema che vuole segnalare l'utente. Prevede come primo step la scelta del progetto/ordine/attività al quale il ticket fa riferimento. Se all'utente è associato un solo progetto/ordine/attività viene utilizzato tale progetto/ordine/attività in automatico.

Scelta l'associazione l'utente può inserire le informazioni utili per risolvere il problema quali:

- tipo di problema,
- descrizione del problema,
- aggiunta di foto relative al problema (fotocamera o galleria),
- aggiunta di altre informazioni per individuare meglio il problema.

Il tipo di problema che può essere inserito dall'utente varia in base al tipo di associazione:

- Progetto,
  - bug (l'app va in errore all'apertura),
  - mancanza (l'app non presenta la visualizzazione delle foto a schermo intero),
  - errore funzionalità (l'app dovrebbe aprire la fotocamera ma apre la galleria),
  - altro.
- Ordine,
  - ordine errato (sono stati ordinati oggetti sbagliati),
  - ordine non ricevuto (l'ordine non è stato consegnato),
  - ordine sbagliato (sono stati recapitati oggetti sbagliati),
  - ordine stato (non si conosce lo stato dell'ordine),
  - altro.
- Attività,
  - attività non eseguita,
  - attività errata,
  - altro.

I problemi descritti dai ticket che presentano tipo "altro" potranno poi essere aggiunti in seguito all'elenco chiuso di quelli disponibili. Tale operazione verrà effettuata da parte degli operatori del sistema se i problemi descritti presentano le caratteristiche per essere rappresentativi di una nuova tipologia di problemi.

2.1.2 Assegnazione di un ticket

L'assegnazione del ticket è una funzionalità prevista solo per l'utente del sistema con ruolo di manager.

Consiste nell'assegnazione

- di una priorità al ticket sulla base di una valutazione del problema,
- di un operatore o un altro manager che dovrà occuparsi del problema.

Il manager può visualizzare i ticket di tutti i clienti, in qualunque stato essi si trovino. L'operatore può visualizzare solo i ticket a lui assegnati (quindi quelli aventi stato in gestione o chiuso) o da lui creati. Il cliente può visualizzare tutti i ticket a lui riferiti.

2.1.3 Gestione di un ticket

La gestione di un ticket prevede la visualizzazione delle informazioni riguardo al ticket inviato insieme allo scambio di messaggi tra operatore ed utente e lo stato del ticket (chiuso, aperto, in gestione).

L'utente, all'interno della gestione del ticket, può inviare un messaggio riguardante il ticket e chiudere il ticket inserendo:

- la motivazione della chiusura (risolto, non risolto, altro),
- la valutazione del servizio offerto,
- il commento al servizio offerto.

Lo scambio di messaggi inizia quando il servizio invia la prima risposta all'utente e prevede l'invio di testo e immagini.

2.1.4 Visualizzazione di un progetto/ordine/attività

La visualizzazione di un progetto, ordine o attività prevede la visualizzazione delle informazioni su tale oggetto e i relativi ticket. Inoltre l'utente potrà aprire direttamente un ticket dalla schermata di visualizzazione dell'oggetto saltando la seleziona iniziale.

2.1.5 Aggiunta di un progetto/ordine/attività/utente

L'aggiunta di un progetto/ordine/attività/utente è riservata agli utenti aventi ruolo di manager od operatore.

Ad eccezione dell'aggiunta di un utente, per compiere le altre azioni è necessario prima scegliere l'utente a cui il progetto/ordine/attività da aggiungere fa riferimento e poi inserire le informazioni obbligatorie a seconda della categoria scelta.

2.1.6 Gestione del profilo

La gestione del profilo prevede la possibilità di modificare le informazioni dell'utente autenticato al sistema come:

- dati anagrafici (nome e cognome),
- dati di contatto (email, telefono, indirizzo),
- dati social (foto profilo, banner profilo).

2.2 Diagramma dei casi d'uso

Attraverso questo diagramma viene presentato, in modo grafico e tramite il formalismo UML, il modo in cui gli attori (utenti) interagiscono con il sistema sopra descritto, senza specificare la logica interna di ogni funzionalità né, tanto meno, la struttura del sistema stesso.

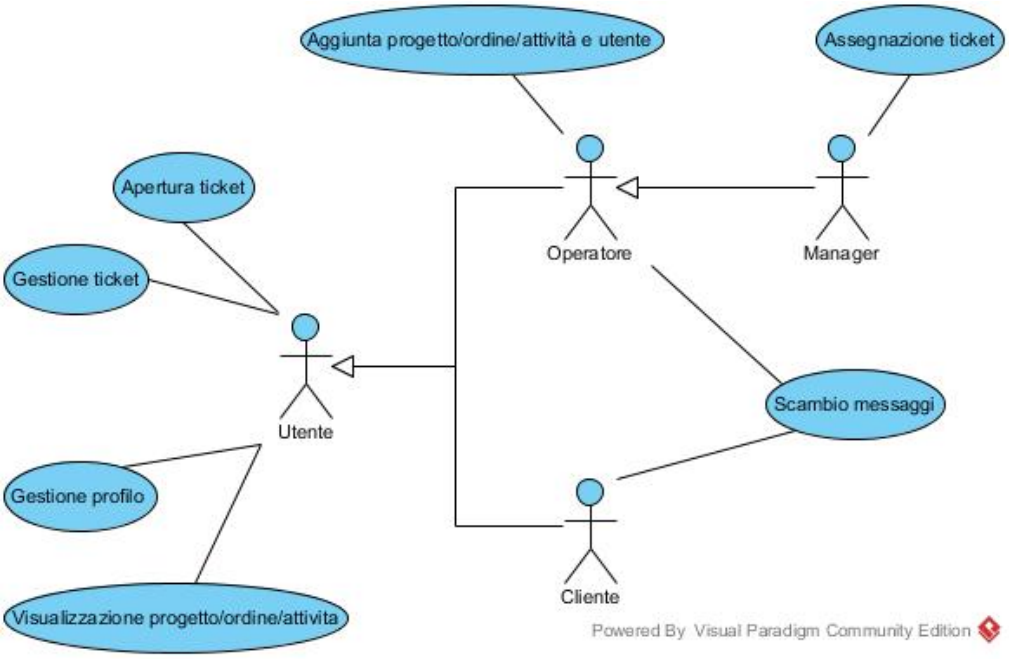


Figura 2.1: Diagramma dei casi d'uso

Gli attori sono rappresentati dal Cliente, l'Operatore e il Manager ed identificano il ruolo che un'entità esterna (Utente) assume quando interagisce con il sistema.

Le funzionalità del sistema sono rappresentate dai casi d'uso e producono dei risultati osservabili dagli attori.

2.3 Sistemi simili già esistenti

Cercando sugli Store dei principali sistemi operativi mobile le applicazioni con lo stesso obiettivo di quella progettata in questa tesi sono numerose ma si differenziano molto tra di loro.

Quelle più scaricate riguardano l'apertura di ticket in merito a problemi generici del mondo IT ed operano come una sorta di community dove gli utenti si aiutano tra di loro per la risoluzione di un problema.

Sulla base delle specifiche richieste nelle sezioni precedenti, quelle più simili individuate sono:

- *ServiceDesk Plus*, si occupa delle funzioni che riguardano la parte Manager ed Operatore del sistema; rispetto al sistema progettato salva in un registro il tempo che il tecnico utilizza per la risoluzione della problematica
- *AlmavivA Mobile Help Desk*, si occupa delle funzioni che riguardano la parte Cliente del sistema; rispetto al sistema progettato fornisce una funzionalità per il tracking del ticket che può basarsi ad esempio sull'id, sull'intervallo di date di quando è stato pubblicato... ma non fornisce la possibile di comunicare con l'operatore attraverso uno scambio messaggi direttamente legato al ticket

Dal momento che nessuna delle applicazioni già esistenti presenta tutte le funzionalità richieste dall'azienda è stato deciso di crearne una nuova.

Capitolo 3

Progettazione

In questo capitolo verrà trattata la progettazione del sistema sulla base dell'analisi effettuata nel capitolo precedente.

3.1 Progettazione concettuale

La progettazione concettuale rappresenta il livello più alto della progettazione di una base di dati.

In questa fase è stato scelto come modello logico secondo il quale strutturare il database il **modello relazionale** e sono stati individuati gli oggetti o entità che costituiscono il sistema e le relazioni o associazioni che li legano. **Entità** e **relazioni** sono due degli elementi fondamentali che compongono il modello **ER**, **Entity/Relationship** utilizzato per rappresentare graficamente lo schema del database.

Prodotto finale di questa fase è lo schema concettuale o schema ER.

3.1.1 Analisi di entità ed associazioni

Utente: entità che rappresenta una generica istanza della persona fisica che utilizza il software, identificata univocamente all'interno del database tramite l'*username*.

La coppia *username* e *password* consente all'utente di autenticarsi al sistema e, sulla base del suo *ruolo*, che può essere Manager, Operatore o Cliente, ha accesso a diverse funzionalità.

Comprende inoltre dati anagrafici (*nome* e *cognome*), dati di contatto (*email*, *telefono*, *indirizzo*) e dati social (*foto_banner* e *foto_profilo*) ma anche i dati relativi alla registrazione presso il servizio che consente lato server l'invio di messaggi al device (*registration_id* e *created_at*) utilizzati per la ricezione delle notifiche push.

All'Utente possono essere associati diversi progetti/ordini/attività attraverso le associazioni **appartentenza_progetto**, **appartentenza_ordine** e **appartenenza_attivita**. Tali associazioni collegano rispettivamente l'entità Utente alle entità Progetto, Ordine ed Attività presentate in seguito. Hanno cardinalità 0 a N lato Utente e cardinalità 1 a 1 lato Progetto/Ordine/Attività. Questo significa che ad un Utente possono essere associati, ad esempio, progetti ed ordini ma non attività ed un progetto/ordine/attività fa riferimento ad un cliente specifico.

Progetto: entità che rappresenta una generica istanza di progetto che la software house si è impegnata a realizzare; questa entità è stata pensata per comprendere tutti i progetti per i quali è stato precedentemente firmato un preventivo da parte del cliente.

Il progetto è identificato univocamente da un id (*id_progetto*), ha un *referente*, un nome (*nome_progetto*) ed una *descrizione* delle specifiche richieste dal cliente.

Ordine: entità che rappresenta una generica istanza di ordine effettuato dal cliente e registrato nel sistema. L'ordine è identificato univocamente da un id (*id_ordine*), ha una data in cui viene effettuato (*data_ordine*), un *oggetto*, un *importo*, una *descrizione* degli articoli, uno stato di spedizione (*stato_spedizione*) e una modalità di pagamento (*metodo_pagamento*).

Attività: entità che rappresenta una funzionalità generica relativa ad un sistema, che la software house si impegna a realizzare per il cliente senza che vi sia nessuna commessa dietro. Dietro mio suggerimento è stato deciso di creare un'entità di questo tipo e non fare confluire tutte queste funzionalità all'interno di un progetto generico perché, per come è pensato il progetto, prevede una commessa.

L'attività è identificata univocamente da un id (*id_attivita*), ha una data in cui viene eseguita (*data_attivita*), un nome (*nome_attivita*) ed una *descrizione*.

Problema: entità che rappresenta una tabella di look-up dove reperire il tipo di problema che ha causato l'apertura del ticket. Al suo interno i problemi sono divisi in categorie per progetti/ordini/attività.

Il problema è identificato univocamente da un id (*id_problema*), ha un nome (*nome_problema*) ed una *categoria* di appartenenza.

Ticket: entità che rappresenta una generica richiesta di assistenza; è il fulcro attorno al quale ruotano tutte le operazioni principali.

Il ticket è identificato univocamente da un id (*id_ticket*), è associato ad un problema attraverso *id_problema*, contiene una *descrizione* ed, opzionalmente, altre informazioni necessarie ad individuare meglio la problematica riscontrata (*altre_info [0-1]*).

L'attributo *stato* è autoesplicativo e può assumere diversi valori, (Aperto, In gestione e Chiuso); *priorità[0-1]* e *assegnato_a[0-1]* non sono valorizzati fino a che il ticket non passa in stato "In gestione" che avviene dopo l'assegnazione del ticket da parte del manager.

L'attributo *aperto_operatore[0-1]* ha natura opzionale perché è valorizzato solo quando il ticket in questione è creato da un operatore.

motivazione_chiusura, *commento_servizio* e *valutazione_servizio* sono attributi che vengono valorizzati solo alla chiusura del ticket e rappresentano il feedback dell'utente in merito al servizio ricevuto.

Ad un ticket è associata una Conversazione che racchiude tutti i messaggi scambiati tra cliente ed operatore. Tale collegamento è realizzato con una associazione chiamata **legato_conversazione**, che ha cardinalità 1 a 1 su entrambi i rami dell'associazione.

Possono essere associati al ticket anche degli allegati contenuti all'interno di Immagine che verrà presentata in seguito. Questo si realizza tramite l'associazione **allegato** che ha cardinalità 0 a 1 lato Immagine e 0 a 5 lato Ticket. La cardinalità 0 a 1 è data dal fatto che la tabella Immagine contiene anche altre immagini che non rappresentano per forza l'allegato di un Ticket. La cardinalità 0 a 5 è stata scelta al posto di quella 0 a N per limitare il numero di allegati che l'utente può inserire alla creazione del Ticket.

Ticket rappresenta una generalizzazione di un gruppo di entità: **TicketProgetto**, **TicketOrdine** e **TicketAttivita**. Insieme costituiscono una gerarchia avente copertura totale ed esclusiva.

TicketProgetto, TicketOrdine e TicketAttivita sono specializzazioni di Ticket ognuna delle quali possiede gli attributi di Ticket e partecipa alle associazioni a cui Ticket è collegato.

Queste specializzazioni sono collegate al progetto/ordine/attività a cui fanno riferimento attraverso le associazioni **legato_progetto**, **legato_ordine** e **legato_attività**. La cardinalità di queste associazioni è 0 a N lato Progetto/Ordine/Attività e 1 a 1 dall'altro lato. Questo significa che ad un progetto/ordine/attività possono essere legati più TicketProgetto/TicketOrdine/-TicketAttivita ed il collegamento del ticket ad una delle tre entità sopracitate

esclude il collegamento alle altre due. Una volta creato un ticket su un progetto, ad esempio, l'associazione che verrà valorizzata sarà legato_progetto mentre non verranno valorizzate le altre due associazioni.

Immagine: entità che rappresenta le immagini che vengono salvate sul server.

Le immagini sono identificate univocamente un da id (*id_immagine*) e possono fare riferimento ad una foto profilo, un banner profilo, ad un allegato di un ticket o di un messaggio scambiato tra il cliente e il servizio.

Ogni immagine ha un *nome* ed un *formato*; alla richiesta di un immagine il path completo relativo a questa verrà costruito lato server ed inviato al client che ne fa richiesta.

Conversazione: entità che rappresenta lo scambio di messaggi che avviene all'interno di un ticket tra il cliente e l'operatore.

La conversazione è identificata univocamente da un id (*id_conversazione*).

L'entità conversazione è collegata tramite l'associazione **composizione** all'entità Messaggio; tale associazione ha cardinalità 0 a N lato Conversazione e 1 a 1 lato Messaggio: questo significa che una conversazione è composta da una serie di messaggi ed ognuno di questi fa riferimento ad una specifica conversazione.

Messaggio: entità che rappresenta il singolo messaggio inviato all'interno di una conversazione.

Il messaggio è identificato univocamente da un id (*id_messaggio*); ha un autore (*username_autore*), un *timestamp* che indica il momento di invio, un *contenuto* o un *allegato*. I campi contenuto ed allegato non possono essere valorizzati contemporaneamente: questo significa che un messaggio può contenere o solo testo o solo un immagine.

3.1.2 Schema concettuale

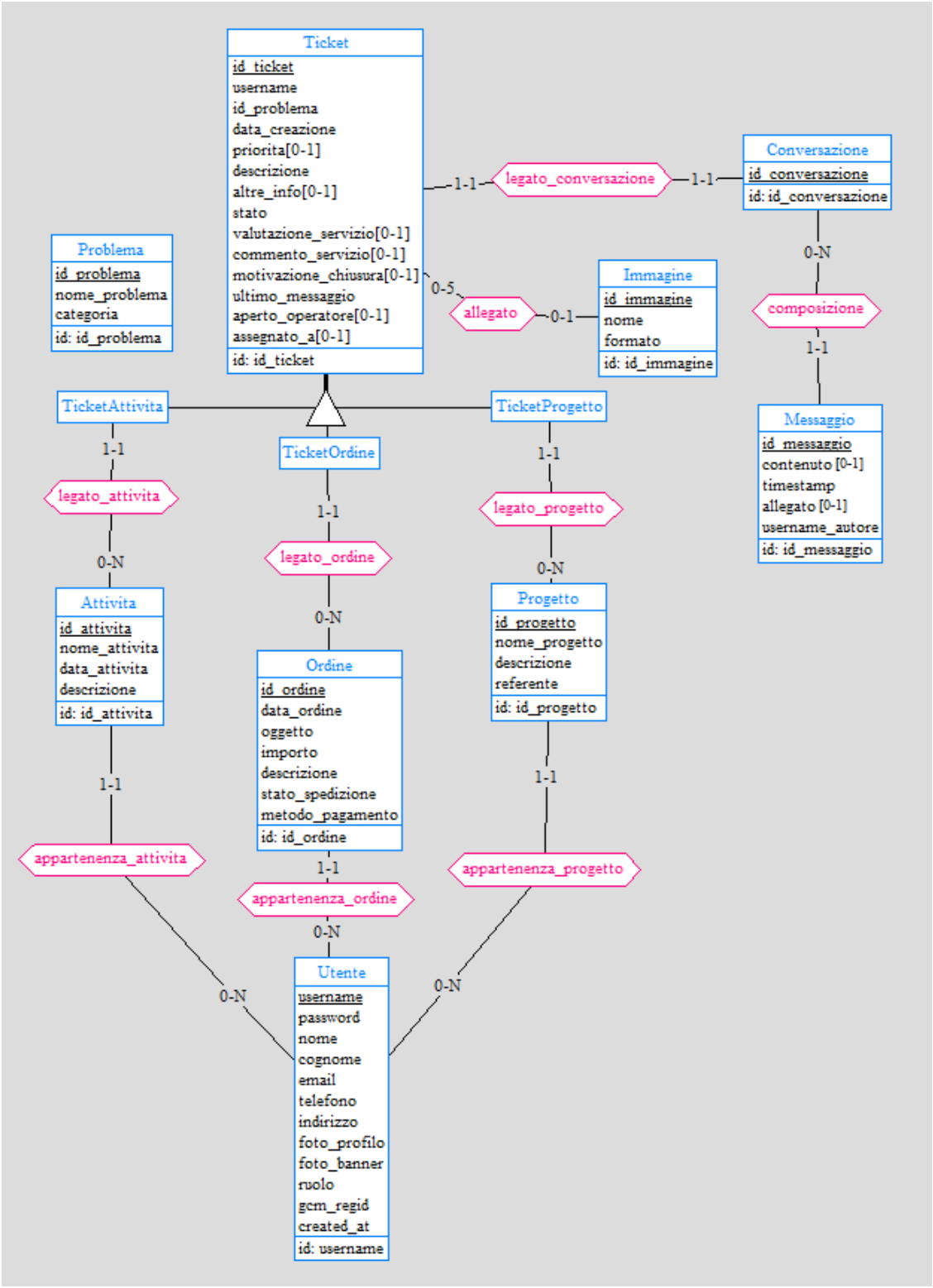


Figura 3.1: Schema ER

3.2 Progettazione logica

La progettazione logica rappresenta il passo successivo alla progettazione concettuale.

In questa fase lo schema concettuale viene tradotto nel modello dei dati del DBMS.

Prodotto finale di questa fase è lo schema logico, espresso nel DDL del DBMS, equivalente allo schema concettuale dal punto di vista della sua capacità informativa.

3.2.1 Eliminazione della gerarchia

Il modello relazionale non può rappresentare direttamente le gerarchie: occorre perciò sostituirle con entità ed associazioni.

Esistono diversi metodi per ottenere questo risultato e la scelta di uno di questi prevede delle considerazioni in merito alla copertura della gerarchia.

Nel caso in questione la gerarchia ha copertura *totale ed esclusiva*. Questo implica la possibilità di utilizzare diversi metodi per la sua traduzione. I metodi presi in considerazione sono il collasso verso il basso e il collasso verso l'alto.

Scegliendo il collasso verso il basso gli attributi dell'entità genitore vengono inseriti in ognuna delle entità figlie e le associazioni connesse all'entità genitore vengono moltiplicate per ognuna delle entità figlie.

Applicando questo metodo allo schema in questione il risultato sarebbe la scomparsa dell'entità Ticket e la modifica di TicketAttività, TicketOrdine e TicketProgetto che, ereditando gli attributi dal genitore, diventerebbero pressoché equivalenti, ad eccezione di un identificativo al loro interno. Ogni associazione connessa all'entità genitore, inoltre, verrebbe trasportata su ognuna delle entità figlie causando un moltiplicarsi di associazioni.

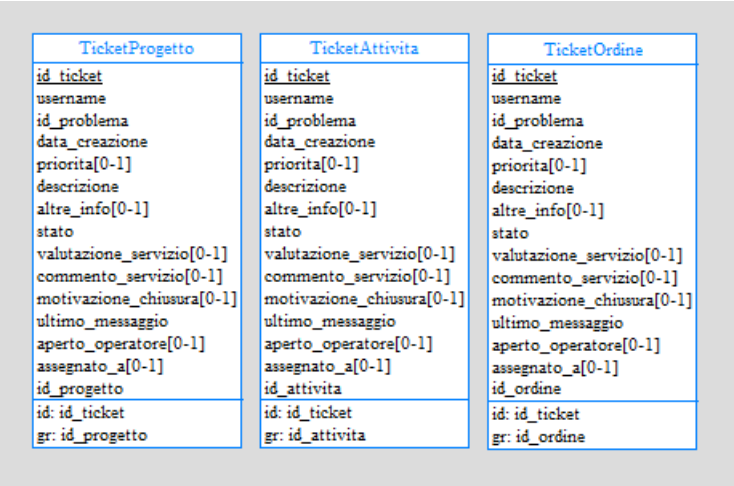


Figura 3.2: Entità figlie nel caso di collasso verso il basso

Per mantenere lo schema concettuale il più semplice possibile questo metodo è stato scartato e la scelta finale è ricaduta sul *collasso verso l'alto*.

Per effetto di questa decisione le entità figlie vengono accorpate nell'entità genitore, le associazioni connesse alle entità figlie vengono trasportate su quella genitore e le cardinalità minime diventano tutte 0.

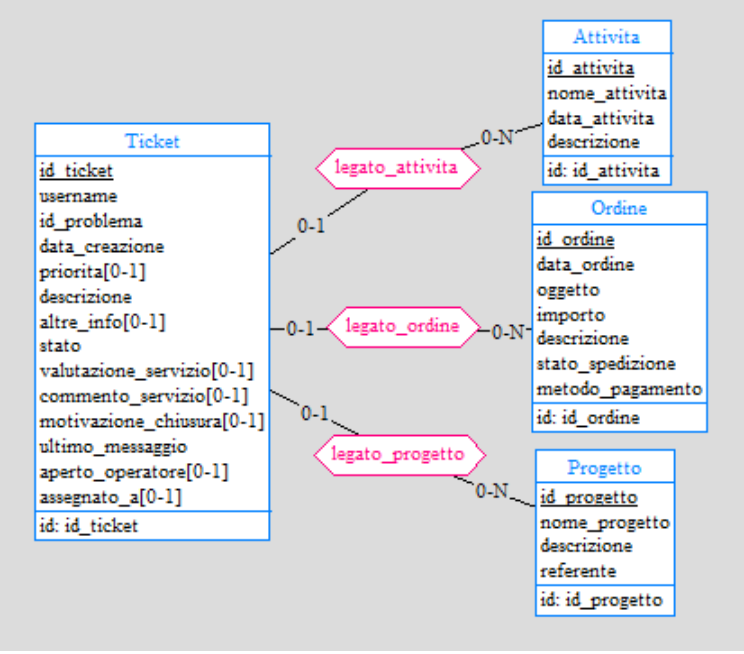


Figura 3.3: Gerarchia dopo il collasso verso l'alto

3.2.2 Traduzione delle associazioni

A questo punto tutte le associazioni presenti all'interno dello schema concettuale sono tutte associazioni binarie uno a uno o uno a molti. Per questo motivo, durante la fase di traduzione, nessuna nuova entità verrà introdotta nello schema (cosa che sarebbe avvenuta in caso di associazioni molti a molti).

Le informazioni relative alle associazioni presenti nello schema verranno inglobate all'interno di una delle entità ai capi dell'associazione considerata.

Segue la traduzione delle associazioni.

allegato (lega Ticket ed Immagine):
Immagine(*id_immagine*, nome, formato, id_ticket[0-1])
FK: id_ticket REFERENCES Ticket

legato_conversazione (lega Ticket e Conversazione):
Ticket(*id_ticket*, username, id_problema, data_creazione, priorita, descrizione, altre_info[0-1], stato, valutazione_servizio[0-1], commento_servizio[0-1], motivazione_chiusura[0-1], ultimo_messaggio, aperto_operatore[0-1], assegnato_a[0-1], id_conversazione)
FK: id_conversazione REFERENCES Conversazione

composizione (lega Conversazione e Messaggio):
Messaggio(*id_messaggio*, timestamp, contenuto[0-1], allegato[0-1], username_autore, id_conversazione)
FK: id_conversazione REFERENCES Conversazione

legato_attivita (lega Ticket e Attivita):
Ticket(*id_ticket*, username, id_problema, data_creazione, priorita, descrizione, altre_info[0-1], stato, valutazione_servizio[0-1], commento_servizio[0-1], motivazione_chiusura[0-1], ultimo_messaggio, aperto_operatore[0-1], assegnato_a[0-1], id_conversazione, id_attivita[0-1])
FK: id_conversazione REFERENCES Conversazione
FK: id_attivita REFERENCES Attivita

legato_ordine (lega Ticket e Ordine):
Ticket(*id_ticket*, username, id_problema, data_creazione, priorita, descrizione, altre_info[0-1], stato, valutazione_servizio[0-1], commento_servizio[0-1], motivazione_chiusura[0-1], ultimo_messaggio, aperto_operatore[0-1], assegnato_a[0-1], id_conversazione, id_attivita[0-1], id_ordine[0-1])
FK: id_conversazione REFERENCES Conversazione
FK: id_attivita REFERENCES Attivita
FK: id_ordine REFERENCES Ordine

legato_progetto(lega Ticket e Progetto):
Ticket(*id_ticket*, username, id_problema, data_creazione, priorita, descrizione, altre_info[0-1], stato, valutazione_servizio[0-1], commento_servizio[0-1], motivazione_chiusura[0-1], ultimo_messaggio, aperto_operatore[0-1], assegnato_a[0-1], id_conversazione, id_attivita[0-1], id_ordine[0-1], id_progetto[0-1])
FK: id_conversazione REFERENCES Conversazione
FK: id_attivita REFERENCES Attivita
FK: id_ordine REFERENCES Ordine
FK: id_progetto REFERENCES Progetto

appartenenza_attivita (lega Attivita ed Utente):
Attivita(*id_attivita*, nome_attivita, data_attivita, descrizione, username)
FK: username REFERENCES Utente

appartenenza_ordine (lega Ordine ed Utente):
Ordine(*id_ordine*, oggetto, data_ordine, importo, descrizione, stato_spedizione, metodo_pagamento, username)
FK: username REFERENCES Utente

appartenenza_progetto (lega Progetto ed Utente):
Progetto(*id_progetto*, nome_progetto, descrizione, referente, username)
FK: username REFERENCES Utente

3.2.3 Schema logico

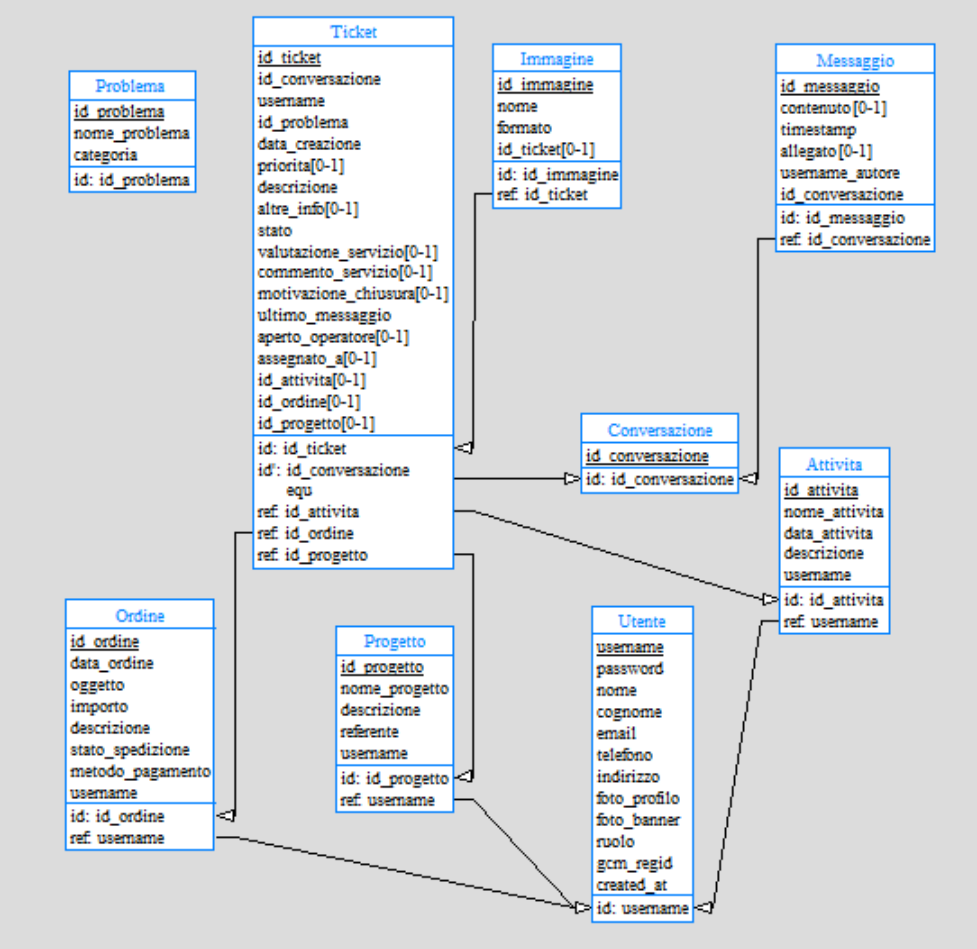


Figura 3.4: Schema logico

3.3 Dati derivati

Un dato derivato è un'informazione derivabile, in modo diretto o tramite relazioni, dalle altre informazioni presenti all'interno del database.

Nello schema logico in figura precedente si può notare la presenza, all'interno dell'entità Ticket, dell'attributo username. Tale informazione, che indica l'appartenenza del ticket ad uno specifico cliente, si sarebbe potuta ricavare navigando lo schema da Ticket a Progetto/Ordine/Attività a seconda della foreign key valorizzata.

A livello di interrogazione del database, visualizzare i ticket di uno specifico cliente, in mancanza di questo attributo, avrebbe comportato la consultazione della tabella Progetto/Ordine/Attività e della tabella Ticket per ogni Progetto/Ordine/Attività associato al cliente. In presenza di questo attributo, invece, basta consultare la sola tabella Ticket (nessun join è necessario).

Dal momento che la visualizzazione dei ticket è una delle funzionalità principali su cui si fonda l'applicazione, ho ritenuto vantaggioso inserire in Ticket l'attributo username.

Sempre all'interno dell'entità Ticket è presente anche l'attributo ultimo_messaggio. Tale informazione è autoesplicativa ed indica appunto il timestamp dell'ultimo messaggio inviato o ricevuto all'interno della conversazione collegata al ticket.

In fase implementativa è stato scelto di dare la possibilità di filtrare la lista di ticket collegati all'utente anche in base all'ultimo messaggio ricevuto. Come si vedrà in seguito, ma è stato già accennato, l'applicazione non mantiene alcun dato salvato in locale, quindi recuperare la data dell'ultimo messaggio comporterebbe risalire ad ogni messaggio della conversazione collegata al ticket per scegliere quello inviato in data più recente.

Dal momento che in fase di scaricamento i ticket vengono ogni volta ordinati in base a questo criterio, ho ritenuto vantaggioso inserire in Ticket l'attributo ultimo_messaggio.

3.4 Web Server

Come anticipato nel primo capitolo, il database progettato è funzionale all'applicazione. Questo significa che, per ottenere i dati necessari al suo funzionamento dovrà richiamare dei servizi del web server che gli restituiranno i dati come risposta.

Suddividendoli sulla base dell'analisi delle principali funzionalità, esposta nel capitolo precedente, sono stati individuati i seguenti servizi principali:

- *login*, per autenticarsi al sistema

Apertura di un ticket

- *getProblemi*, per ottenere la lista chiusa dei problemi relativi alla categoria del ticket che si vuole aprire

- *addTicket*, per aggiungere il ticket al database

Assegnazione di un ticket

- *assignTicket*, per assegnare priorità ed operatore ad un ticket aperto

Gestione di un ticket

- *getTicket*, per ottenere la lista dei ticket
- *getTicketFromId* per ottenere lo specifico ticket
- *chiudiTicket*, per chiudere il ticket
- *getConversazione*, per ottenere lo scambio di messaggi relativo al ticket
- *sendMessages*, per l'invio di messaggi all'interno della conversazione

Visualizzazione di un progetto/ordine/attività

- *getProgetti*, per ottenere la lista dei progetti associati ad un cliente
- *getProgetto*, per ottenere il dettaglio del progetto
- *getTicketOfProgetto*, per ottenere i ticket dello specifico progetto

Aggiunta di un progetto/ordine/attivita/utente

- *addProgetto*, per aggiungere un progetto
- *addUtente*, per aggiungere un utente

Gestione del profilo

- *getUtente*, per ottenere i dati dell'utente
- *modificaInfo*, per modificare le informazioni di contatto
- *modificaPassword*, per modificare la password
- *modificaImmagine*, per modificare l'immagine profilo o copertina

Dei servizi illustrati per i progetti esisteranno gli equivalenti per gli ordini e le attività.

3.5 Mockup

In questa sezione si procede alla progettazione delle schermate principali che verranno utilizzate nell'applicativo in modo da mettere in evidenza il flusso di navigazione da una schermata all'altra dell'applicazione.

Questo avviene tramite l'utilizzo dei mockup, abbozzi di schermate, che, in quanto tali, non rappresentano fedelmente il contenuto del layout grafico ma servono a dare un'idea di massima di come i dati verranno presentati all'utente che utilizza l'applicazione.

Già in questa fase si è cercato di mantenere un'interfaccia semplice ed intuitiva utilizzando una serie di regole grafiche e convenzioni consolidate che permettono all'utente, con un colpo d'occhio, di comprendere come dovrà utilizzare l'applicazione.

Le prossime sezioni illustrano come accedere alle funzionalità descritte in fase di analisi.

Login

La schermata di login è la prima schermata che si incontra all'avvio dell'applicazione.

Essenziale ed intuitiva, contiene il nome dell'applicazione e permette l'autenticazione al sistema che deve avvenire tramite l'inserimento di username e password. In caso di esito positivo l'utente verrà indirizzato alla pagina principale dell'applicazione.

Una volta che l'utente avrà effettuato il login, ai successivi avvii dell'applicazione questa schermata non dovrà essere visualizzata e l'utente dovrà essere reindirizzato alla schermata principale dell'applicazione. Tale schermata tornerà accessibile al logout dall'applicazione per permettere il login di un nuovo utente.

In fase di implementazione è bene prevedere in questa schermata anche il recupero della password da parte dell'utente.

Figura 3.5: Mockup schermata login

Pannello di navigazione

Effettuato l'accesso la schermata principale che l'utente visualizza è diversa a seconda del suo ruolo all'interno del sistema. In entrambi i casi, però, è legata ad un pannello di navigazione laterale destinato a contenere i comandi per la navigazione dell'applicazione.

Se l'utente che ha effettuato l'accesso ha ruolo di Cliente il pannello di navigazione è composto da un header, che fornisce le informazioni principali sul cliente, e dai comandi per accedere alla schermata dei ticket, dei progetti/ordini/attività a lui associati e al suo profilo utente.

Se l'utente che ha effettuato l'accesso ha ruolo di Manager od Operatore il pannello di navigazione è composto da un header come il precedente e dai comandi per accedere alla schermata dei clienti e al suo profilo utente. Scelto il cliente del quale si vogliono occupare, possono accedere allo stesso pannello visualizzato dal cliente al login, con l'integrazione di alcune funzionalità relative al loro ruolo di impiegati.

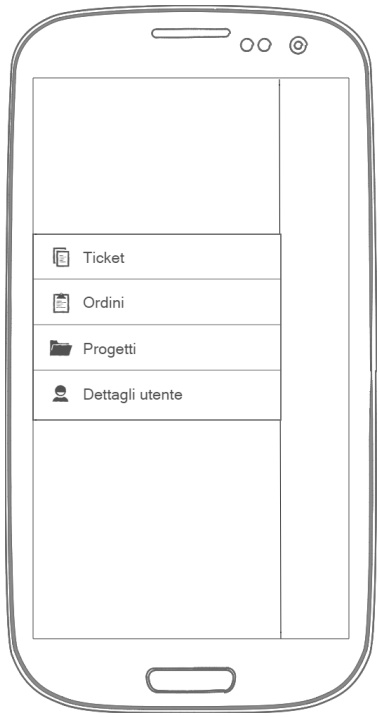
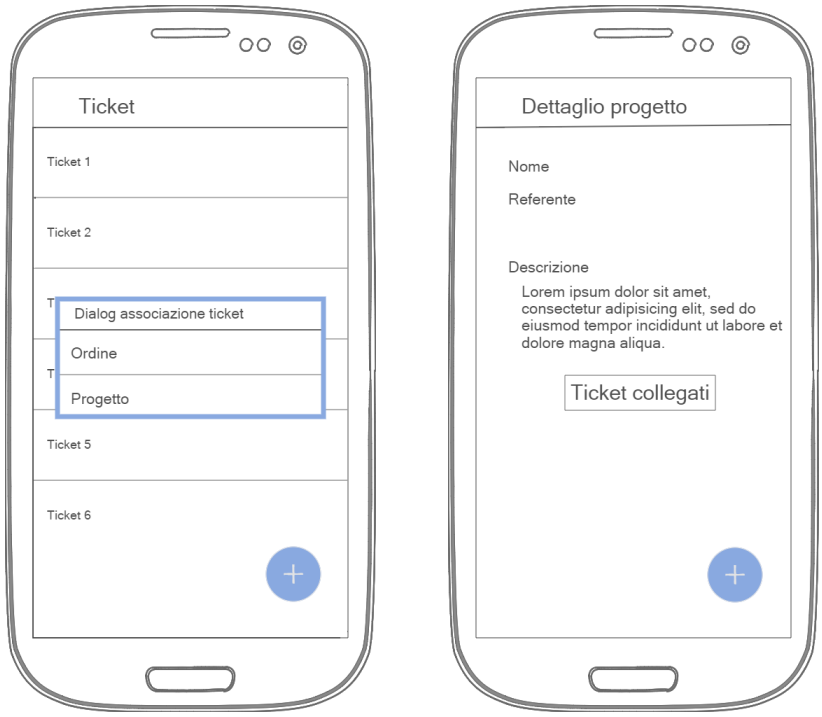


Figura 3.6: Mockup pannello di navigazione Cliente

Apertura di un ticket

L'apertura di un ticket può avvenire dalla schermata principale mostrata al cliente subito dopo il login, che contiene una lista dei ticket che gli appartengono, oppure dal dettaglio di un progetto/ordine/attività.

All'apertura l'utente deve scegliere prima la problematica riscontrata e poi spostarsi in una nuova schermata per inserire tutte le altre informazioni necessarie alla sua individuazione e risoluzione. In questo modo si attribuisce importanza alla scelta del problema perché è la prima cosa che l'Operatore o il Manager guardano all'apertura di un nuovo ticket e sulla base del quale possono già farsi un'idea in merito ai tempi di risoluzione.



(a) Lista ticket (b) Dettaglio progetto

Figura 3.7: Mockup apertura di un ticket

Assegnazione di un ticket

La schermata di assegnazione di un ticket è raggiunta da quella di dettaglio del ticket ed al suo interno vengono inserite le informazioni in merito a priorità e operatore che si deve occupare del ticket. Il pulsante per raggiungere tale schermata è nascosto ad operatori e clienti.

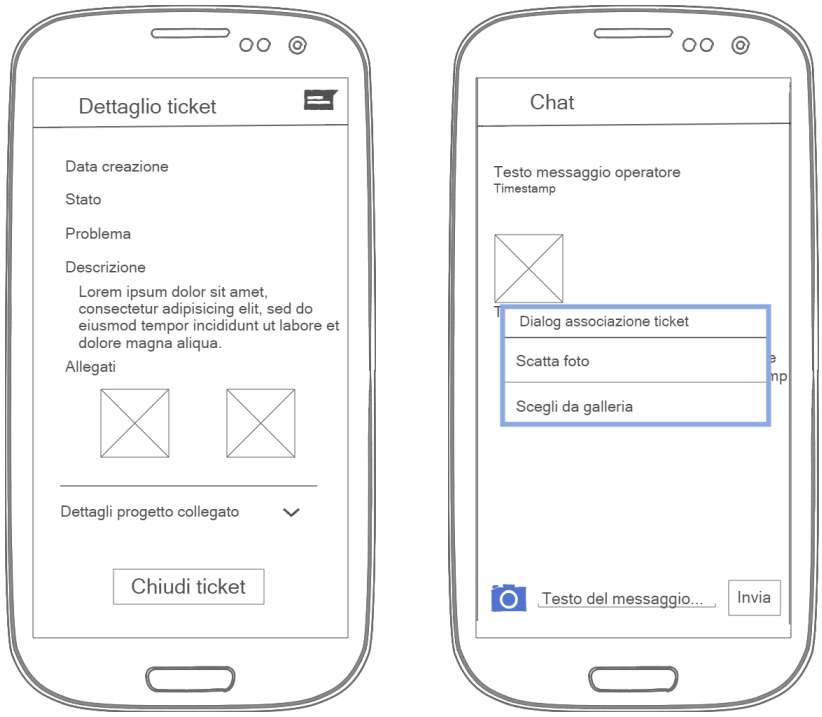
La schermata di dettaglio del ticket è presentata nella sezione successiva.

Gestione di un ticket

La schermata di gestione del ticket si raggiunge tramite la selezione di un ticket dalla schermata principale.

Contiene le informazioni relative al dettaglio del ticket e i pulsanti che permettono di accedere alla schermata relativa

- allo scambio di messaggi con l'operatore che si occupa del servizio (in alto a destra),
- alla chiusura del ticket.



(a) Dettaglio ticket (b) Schermata chat

Figura 3.8: Mockup gestione di un ticket

Visualizzazione di un progetto/ordine/attività

La schermata di dettaglio di un progetto/ordine/attività si può visualizzare selezionando il progetto/ordine/attività dalla schermata contenente la lista di uno dei tre oggetti, raggiungibile dal menù di navigazione laterale.

Contiene le informazioni in merito all'oggetto selezionato, un pulsante che rimanda ad una schermata contenente i ticket a questo collegati e, come anticipato prima, un pulsate per l'aggiunta di un nuovo ticket.

Aggiunta di un progetto/ordine/attività/utente

Le schermate che permettono l'aggiunta di un progetto/ordine/attività sono raggiungibili tramite il menu di navigazione laterale disponibile all'operatore dopo la scelta del cliente.

Gestione del profilo

Attraverso la schermata di gestione del profilo l'utente può modificare dettagli utente, password e immagini profilo e copertina. La pressione di uno dei pulsanti sotto l'header rimanda alle schermate che gestiscono queste funzionalità.

In questa schermata è previsto anche il logout dal sistema.

Figura 3.9: Mockup gestione del profilo

Capitolo 4

Implementazione

In questo capitolo verrà discussa la scelta del sistema operativo e dell'ambiente di sviluppo utilizzato per realizzare il software e verranno trattate le scelte implementative riguardanti la parte Server e la parte Client dell'applicazione.

Il processo di sviluppo si è svolto procedendo di pari passo lato Server e lato Client mano a mano che venivano integrate nuove funzionalità al sistema. Per esigenze organizzative nella presente tesi verranno illustrate in maniera successiva, partendo dalla parte Server.

4.1 Server

Il lato server del sistema ha un ruolo cruciale. Come verrà esposto in seguito, l'applicazione momentaneamente non salva alcun dato in locale, perciò il mancato funzionamento del Server comporta l'impossibilità da parte dell'utente di utilizzare l'applicazione.

4.1.1 Strumenti utilizzati

Il DBMS utilizzato nella seguente tesi è **MySql**, il più diffuso database Open Source basato sul linguaggio SQL. MySql è un RDBMS, ovvero un sistema di gestione per database relazionali, come quello progettato in questa tesi per il quale è stato impostata la collation (codifica dei caratteri delle tabelle) ad UTF-8[1].

[1]UTF-8 è una codifica di caratteri basata su Unicode che utilizza un numero variabile di byte per la codifica dei caratteri.

Il linguaggio **SQL** (Structured Query Language) non è solo un semplice linguaggio di interrogazione (Data Query Language) ma è progettato anche per creare e modificare gli schemi del database (Data Definition Language), inserire, modificare, aggiornare, cancellare i dati presenti su questo (Data Manipulation Language) e creare e gestire strumenti di controllo ed accesso ai dati (Data Control Language).

I linguaggi di programmazione che supportano MySql sono numerosi e tra questi è stato scelto PHP. **PHP** è un linguaggio di scripting server-side che permette l'elaborazione di dati da database MySQL attraverso l'utilizzo di opportune query SQL. Tutto ciò è possibile grazie ad alcune estensioni, tra cui spicca per importanza MySQLi che risulta essere al momento l'estensione più performante e dinamica.

4.1.2 Pattern di comunicazione tra Client e Server

Pattern di richiesta lato Client

Per comunicare con il server l'applicazione utilizza il protocollo HTTP e POST come metodo di invio dei dati. Per effetto dell'utilizzo di POST i parametri arrivano nel corpo della richiesta HTTP.

Dal momento che l'implementazione della richiesta lato client è legata al sistema operativo mobile utilizzato, un esempio in merito verrà presentato in seguito.

Pattern di risposta lato Server

Per consentire la comunicazione tra il Server e l'applicazione è stato definito un pattern di risposta che produce una risposta in **JSON** (JavaScript Object Notation), un formato di testo completamente indipendente dal linguaggio di programmazione.

Tale caratteristica, insieme ad altre, rende il linguaggio JSON ideale per lo scambio di dati [3].

Ogni risposta inviata dal server segue questo schema

```
{
   "success": true|false,
   "message": null,
   "data": null
}
```

Listato 4.1: Pattern risposta lato server

dove:

- *success* contiene l'esito della chiamata al server,
- *message* contiene il testo relativo all'esito positivo della chiamata o relativo all'errore in caso di esito negativo,
- *data* contiene l'oggetto con i dati restituiti dal server, null se la chiamata non restituisce alcun dato.

4.1.3 Esempi di risposta servizi lato server

A seconda dei dati richiesti, come valori del campo *data* nella risposta, possono essere utilizzati oggetti json od array json.

Di seguito vengono riportati alcuni esempi di risposta che utilizzano oggetti json ed array json e che danno modo di illustrare alcune scelte implementative riguardanti il database e il lato server.

Login

Il servizio *login.php* viene viene richiamato quando l'utente vuole autenticarsi al sistema.

I parametri passati in POST sono username e password.

Una volta individuato l'utente sul database, la password inserita, codificata in **SHA256 (Secure Hash Algorithm)**, viene confrontata con quella salvata sul database in SHA256.

SHA256 è una funzione di hash e, come tutte le funzioni di hash,

- è irreversibile,
- prende in input una stringa di lunghezza variabile e ne produce una di lunghezza fissa.

Prima di scegliere questa funzione di hash, della famiglia SHA-2, è stata presa in considerazione anche MD5 (Message Digest 5). Mentre la stringa prodotta in output da MD5 è di 128 bit e richiede 16 bit di spazio di archiviazione, quella prodotta da SHA256 è di 256 bit e richiede 32 bit di spazio di archiviazione. Questo significa che la possibilità che si verifichino collisioni, ovvero due stringhe in output identiche, utilizzando SHA256 è molto minore rispetto a MD5 (esattamente 2^{256} / 2^{128} = 2^{128} in meno) [4].

Per questo motivo è stato scelto di utilizzare SHA256 rispetto ad MD5.

Il vantaggio di usare gli hash salta subito all'occhio: se un malintenzionato riesce ad accedere al database progettato, troverà solo gli hash delle password e non le password in chiaro.

Il servizio restituisce, in caso di successo:

```
{
    "success": true,
    "message": "Login cliente effettuato",
    "data": {
        "username": "annagiulia",
        "ruolo": "Cliente"
    }
}
```

Listato 4.2: Risposta positiva servizio login.php

In questo caso *data* è costituito da un json object che contiene l'esito del login, e il ruolo dell'utente loggato.

Il servizio restituisce, in caso di errore:

```
{
    "success": false,
    "message": "Errore di login",
    "data": null
}
```

Listato 4.3: Risposta negativa servizio login.php

In questo caso *data* è null. Il messaggio non cambia se si vuole accedere con un username inesistente o se si sbaglia la password.

Get utente

Il servizio *get-utente.php* viene viene richiamato ogni volta che l'applicazione richiede i dati dell'utente loggato o dell'utente che Operatore o Manager vogliono visualizzare.

L'unico parametro passato in POST è l'username.

Il servizio restituisce, in caso di successo:

```
{
    "success": true,
    "message": "Utente scaricato correttamente",
    "data": {
        "username": "annagiulia",
        "nome": "Anna Giulia",
        "cognome": "Leoni",
        "email": "annagiulia.leoni@hotmail.it",
        "telefono": "3496247023",
        "indirizzo": "Via Silvio Corbari, 4 - Santo Stefano - RA",
        "ruolo": "Cliente",
        "immagine_profilo": "http://nome_del_dominio/php/uploads/
            8eca0ee7e8a93238fe4c187f5f2b319c.jpg",
        "banner_profilo": ""
    }
}
```

Listato 4.4: Risposta servizio get-utente.php

In questo caso *data* è costituito da un json object che contiene tutti i dati relativi all'utente per il quale si è fatto richiesta.

Come è possibile notare, i campi relativi ad *immagine_profilo* e *banner_profilo* contengono il path finale da cui effettuare il download dell'immagine, se presente, nulla altrimenti. Il path dove risiede il file all'interno del file system del server, viene costruito lato server, recuperando nome ed estensione dell'immagine dal database, prima di inviare la risposta in json.

La stessa operazione viene effettuata ogni volta che il json contiene un'immagine, che essa si tratti di immagine profilo, banner profilo, allegato di un ticket.

Get ticket

Il servizio *get-ticket.php* viene viene richiamato ogni volta che l'applicazione richiede i ticket associati ad uno specifico utente.

L'unico parametro passato in POST è l'username.

Il servizio restituisce, in caso di successo:


```
{
    "success": true,
    "message": "Ticket scaricati",
    "data": [
        {
            "username": "annagiulia",
            "id_ticket": 1,
            "id_attivita": null,
            "id_ordine": null,
            "id_progetto": 1,
            "data_creazione": "10/07/2015 12:44:23",
            "priorita": "Bassa",
            "nome_problema": "Mancanza",
            "categoria": "Progetto",
            "descrizione": "Non funziona il login",
            "altre_info": "",
            "nome_stato": "Chiuso",
            "valutazione_servizio": 3,
            "commento_servizio": "Considerato il problema mi
                aspettavo venisse preso in carica
         prima. In generale solo soddisfatta del servizio fornito",
            "motivazione_chiusura": "Risolto",
            "id_conversazione": 1,
            "ultimo_messaggio": "12/10/2015 12:44:23",
            "aperto_operatore": "",
            "assegnato_a": "operatore1"
        }
    ]
}
```


Listato 4.5: Risposta servizio get-ticket.php

In questo caso *data* è costituito da un json array che contiene i ticket associati all'utente per il quale si è fatto richiesta.

Come è possibile notare soltanto uno tra *id_progetto*, *id_ordine*, *id_attivita* è valorizzato mentre gli altri due sono valorizzati a null. La valorizzazione a null sul database è stata scelta perché, in fase implementativa lato server, gli script che si occupano di ottenere la lista di progetti/ordini/attività associati all'utente operano rispettivamente sulle condizioni "WHERE id_progetto IS NOT NULL", "WHERE id_ordine IS NOT NULL", "WHERE id_attivita IS NOT NULL".

Il ticket portato come esempio, inoltre, ha stato "Chiuso" infatti, *valutazione_servizio*, *commento_servizio* e *motivazione_chiusura* sono valorizzati. Il campo *aperto_operatore* risulta vuoto, perciò il ticket in questione è stato aggiunto dal Cliente.

4.2 Client

4.2.1 Sviluppo di un'app nativa

In base alle tecnologie su cui si basano, le applicazioni mobile possono essere classificate in [5]

- **app native**: sviluppate con codici e librerie proprietarie, vengono pubblicate e distribuite, gratuitamente o a pagamento, tramite gli Store che garantiscono loro visibilità, diffusione e guadagno; altre caratteristiche che le contraddistinguono sono accesso all'hardware e al software installato nel dispositivo (fotocamera, file system...), ottime prestazioni e un funzionamento off-line nativo.

- **web app**: pagine web che permettono di simulare l'aspetto delle interfacce proprie di app native utilizzando tecnologie Web (HTML5, JavaScript e CSS3); applicazioni di questo tipo non possono essere pubblicate nello Store, non possono interagire con l'hardware e il software del dispositivo, hanno tempi di sviluppo decisamente più bassi rispetto alle app nativa e funzionano solo in presenza di connettività Internet.

- **app ibride**: rappresentano un compromesso tra le due tipologie precedenti perché sono scritte con tecnologie Web ma vengono eseguite localmente all'interno di un'applicazione nativa; alcune caratteristiche sono una minore efficienza nel rendering grafico e una potenziale lentezza di esecuzione nell'accesso alle risorse locali.

Alla luce delle funzionalità richieste dal sistema progettato in questa tesi, la scelta è ricaduta sulle app native.

Le web app sono state scartate a causa della loro impossibilità di interazione con l'hardware e il software del dispositivo (mentre l'applicazione richiesta prevede l'utilizzo della fotocamera e l'accesso alla galleria).

Le app ibride sono state scartate perché, a parità di funzionalità, offrono prestazioni meno elevate delle app native ed inoltre le funzionalità di interfacciamento alle risorse locali dipendono molto dal framework utilizzato e potrebbero non essere complete o non supportate per alcune piattaforme.

4.2.2 Android come sistema operativo mobile

Come è possibile notare dal diagramma sottostante i sistemi operativi attualmente più diffusi sul mercato sono, nell'ordine, Android, iOS e Windows Phone [6].

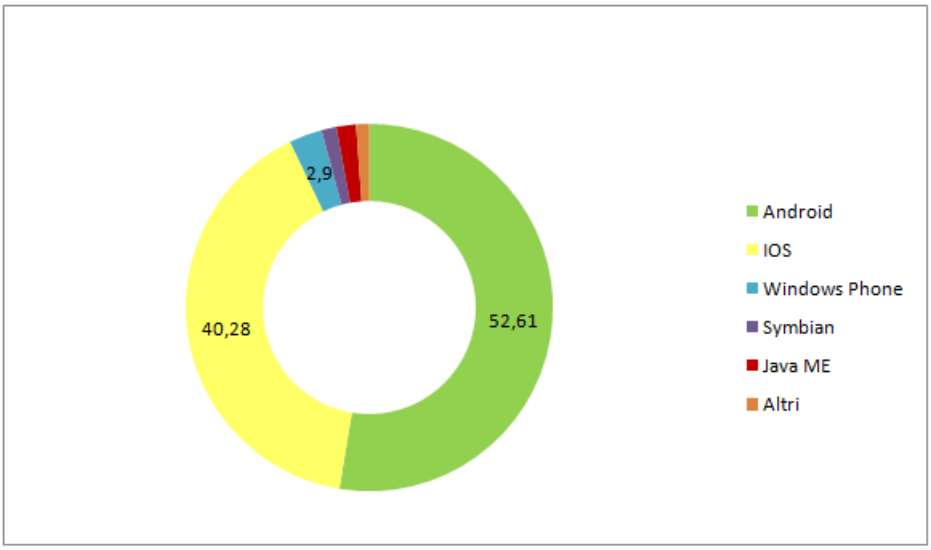


Figura 4.1: Diffusione dei principali sistemi operativi mobile Ottobre 2015

Android

Android è un sistema operativo sviluppato da Google Inc. basato su Kernel Linux.

È per la quasi totalità Open Source (ad esclusione di alcuni driver e alcune google apps incluse come il Google Play store), ed è distribuito sotto i termini della Licenza Libera Apache 2.0. che consente agli utenti di utilizzare, modificare e distribuire versioni modificate del software [7].

Le applicazioni Android sono Java-based e l'ambiente di sviluppo più utilizzato per sviluppare applicazioni Android è Android Studio. L'attuale nonché ultima versione di Android è la 6.0 (Android Marshmallow) e lo store ufficiale attraverso cui vengono distribuite le applicazione è il Google Play Store.

Android è il sistema operativo più diffuso tra i sistemi operativi mobile. Il suo successo è determinato da diversi fattori tra cui la sua adozione anche su dispositivi di fascia bassa appartenenti a diversi produttori e la possibilità di scegliere il dispositivo che più aggrada l'utente in base alle diverse caratteristiche hardware e all'interfaccia grafica che contraddistingue i produttori.

Tuttavia, gli aggiornamenti ufficiali rilasciati da Google per i propri dispositivi, devono essere recepiti dai vari produttori che spesso aggiornano solo i dispositivi più recenti a causa dei costi di aggiornamento.

iOS

iOS è un sistema operativo sviluppato da Apple per iPhone, iPod touch e iPad. Come Mac OS X, è una derivazione di UNIX e usa un microkernel Mach basato su Darwin OS [8].

L'ambiente di sviluppo utilizzato per sviluppare applicazioni iOS è Xcode, disponibile solamente per il sistema operativo Mac OS X. L'attuale versione di iOS è la 9.0 e lo store ufficiale attraverso cui vengono distribuite le applicazioni è l'App Store.

Fattore di successo di iOS, che lo rende il secondo sistema operativo mobile per diffusione, è sicuramente l'elevato grado di coinvolgimento nei confronti dell'utente, e la sua facilità d'uso, che fanno passare in secondo piano il prezzo dei dispositivi.

Windows Phone

Windows Phone è una famiglia di sistemi operativi per smartphone di Microsoft.

L'ambiente di sviluppo utilizzato per sviluppare applicazioni Windows Phone è Visual Studio. L'attuale versione di Windows Phone è la 8.1 e lo store ufficiale attraverso cui vengono distribuite le applicazioni è il Windows Phone Store.

Il problema principale per gli utenti di questo sistema operativo è da sempre la presenza di un minore numero di applicazioni presenti nel suo store rispetto ai principali concorrenti, dovuta al tardo arrivo del sistema sul mercato in confronto a questi ultimi; tale divario si è assottigliato nel corso degli ultimi anni con l'arrivo di quasi tutte le principali applicazioni del mondo mobile [9].

Alla luce di quanto esposto, la scelta del sistema operativo su cui implementare il software è ricaduta su **Android**.

Oltre ad essere il sistema operativo mobile più diffuso e accessibile, anche a coloro che optano per smartphone di fascia bassa, la presenza di una vastissima community di sviluppatori, tutorial, documentazioni, esempi, ha contribuito a prendere una decisione in questo senso.

L'applicazione sviluppata in questa tesi sarà disponibile per i dispositivi che utilizzano una versione di Android superiore od uguale alla **4.0 (Ice Cream Sandwich - API 14)**, in modo da poter esser compatibile con un ampio numero di dispositivi.

4.2.3 Android Studio come ambiente di sviluppo

L'ambiente di sviluppo utilizzato per la realizzazione di questa applicazione è Android Studio, che, andando a sostituire Eclipse e gli ADT (Android Developer Tools), è diventato l'IDE primario di Google per lo sviluppo nativo di applicazioni Android.

Android Studio è caratterizzato dalla presenza del **Gradle**, un sistema di build molto avanzato, che consente, ad esempio, di dichiarare al suo interno le dipendenze del progetto che si sta sviluppando, o ancora offrire diverse versioni di una stessa app all'interno del Google Play Store, una free ed una a pagamento, generando APK[2] multipli da un singolo modulo [10].

4.2.4 Modalità di utilizzo

La modalità di utilizzo online, come già accennato precedentemente, è l'unica disponibile al momento. Il dispositivo non mantiene alcun dato in locale perciò i dati da visualizzare in ogni schermata dell'applicazione vengono richiesti ogni volta al server.

Questo significa che, in assenza di connettività

- i clienti non potranno aprire nuovi ticket, né tantomeno visualizzare quelli già presenti e i progetti/ordini/attività loro associati
- gli operatori, oltre alle operazioni descritte al precedente punto, non potranno cambiare lo stato dei ticket quando iniziano ad occuparsi del problema segnalato oppure aggiungere progetti/ordini/attività relativi ad un cliente

L'introduzione di una modalità offline sarebbe utile ad entrambe le parti almeno per fornire le funzioni che riguardano la visualizzazione di ticket e progetti/ordini/attività associati all'utente.

L'utilizzo della sola modalità online rappresenta un limite di questa applicazione ma il supporto per la modalità offline verrà sicuramente aggiunto in futuro anche per una esigenza di ricostruzione dello stack delle activity al click su una notifica ricevuta.

[2]Un file in formato APK rappresenta il risultato finale della compilazione ed è utilizzato per la distribuzione e l'installazione di componenti in dotazione sulla piattaforma per dispositivi mobili Android.

4.2.5 Suddivisione in moduli

Sull'idea di separare la struttura dei dati veri e propri dalla logica di funzionamento del sistema, secondo il pattern **MVC (Model-View-Controller)**, l'applicazione sviluppata è stata divisa in moduli.

app

Contiene la logica di funzionamento (activity e fragment) e le risorse dell'applicazione (layout, drawable, stringhe, colori...).

Se inquadrato all'interno di MVC, il modulo app rappresenta il Controller e la View.

La classe activity infatti, pur non estendendo la classe View di Android, si occupa di gestire la visualizzazione di una schermata per l'utente e di gestire gli eventi di quella schermata (onCreate, onResume...) operando da View-Controller. La View vera e propria è rappresentata invece dai layout e dalle altre risorse grafiche utilizzate dall'applicazione.

core

Contiene la logica di interfacciamento al server e la struttura dei dati utilizzati dall'applicazione scaricati dal database.

Se inquadrato all'interno di MVC, il modulo core rappresenta il Model.

Si compone di due package:

- *data*, contiene le classi che definiscono la struttura dei dati scaricati dal database

  Di seguito, come esempio, viene riportata la classe Progetto.

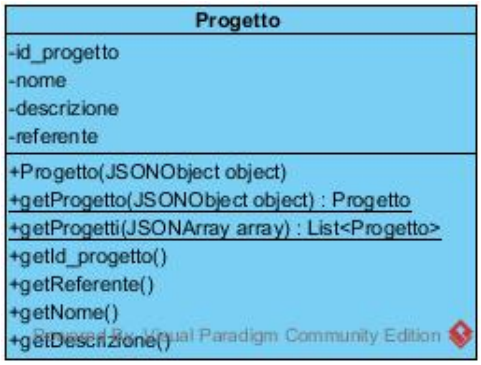


Figura 4.2: UML classe Progetto

All'interno della classe sono presenti, oltre ai getter dei campi privati e al costruttore, due metodi statici volti a creare un'istanza della classe a partire da un JSONObject e una lista di istanze della classe a partire da un JSONArray, utilizzati alla ricezione di dati dal server.

Tutte le altre classi all'interno di *data* sono implementate allo stesso modo.

- *utils*, contiene le classi ServerUtils e ServerResponse, che vengono illustrate con un esempio nella sezione successiva, relativa all'esempio di richiesta dati al server

moduli di librerie

Contengono le librerie esterne utilizzate nello sviluppo dell'applicazione, ad eccezione di quelle che sono state aggiunte come dipendenze al Gradle di app. Alcune sono state fornite dall'azienda, altre trovate all'interno di community di sviluppatori Android che mettono a disposizione funzionalità da loro sviluppate sotto diversi tipi di licenza. Segue un elenco delle principali:

- Network, illustrata nella sezione successiva
- MaterialEditText, MaterialDialog, FloatingActionButton che permettono di costruire interfacce grafiche uniformi secondo il MaterialDesign
- ButterKnife che consente il bind di campi e metodi delle view di Android in modo molto più veloce

4.2.6 Esempio di richiesta dati al server

La richiesta HTTP viene eseguita all'interno dell'applicazione utilizzando una libreria esterna chiamata **Network**. Tale libreria comprende le classi *NetworkSendDataItem* e *NetworkSendDataFile* utilizzate per inviare parametri e file (foto scattate o galleria) in POST. Le due classi sopra citate sono utilizzate all'interno della classe *ServerUtils*.

ServerUtils si compone di una serie di variabili statiche e metodi statici. Le variabili statiche definiscono gli URL dei servizi da richiamare e i nomi delle variabili passati in POST. I metodi statici, che restituiscono un NetworkSendDataItem o un NetworkSendDataFile a seconda del servizio, definiscono la chiamata al servizio.

```
public class ServerUtils {

    /* URL servizi */
   private static String URL_BASE = "http://nome_del_dominio/php/";
   private static final String URL_LOGIN = URL_BASE + "login.php";

   /* Nomi delle variabili in POST */
   private static final String USERNAME = "username";
   private static final String PASSWORD = "password";

   /* Implementazione servizi */
   public static NetworkSendDataItem getLogin(String username,
       String password) {
        NetworkSendDataItem item = new NetworkSendDataItem(URL_LOGIN,
            NetworkItem.NetworkMethod.POST);
        item.putItem(USERNAME, username);
        try {
            item.putItem(PASSWORD, Utils.codificaSHA256(password));
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return item;
    }
}
```

Listato 4.6: Classe ServerUtils

Un'altra classe essenziale in questa fase è *ServerResponse*.

ServerResponse è una classe astratta generica che si occupa invece di gestire la risposta ricevuta dal server e di indirizzarla verso i metodi *onError* ed *onSuccess*, anch'essi astratti, a seconda dell'esito della richiesta.

ServerResponse racchiude quindi la struttura dell'algoritmo di smistamento verso i metodi onError e onSuccess ma lascia il compito di implementarli alle classi che vengono da questa generate. Il parametro T viene sostituito con un JSONObject o un JSONArray a seconda dei dati richiesti al server.

In questo modo viene concretizzato il **pattern Template Method**.

Prendendo come esempio il servizio di login, che permette all'utente di autenticarsi al sistema con username e password, viene illustrata di seguito una tipica chiamata ad un servizio.

```
@Override
public void onRequestLogin(String email, String password) {
   new NetworkSendAsyncTask(ServerUtils.getLogin(email, password),
         new ServerResponse<JSONObject>() {
            @Override
            public void onPreExecute() {
               /* mostra dialog di caricamento */
               super.onPreExecute();
            }
            @Override
            public void onPostExecute(String result) {
               /* elimina dialog di caricamento */
                super.onPostExecute(result);
            }
            @Override
            public void onError(String message) {
               /* azione da intraprendere in caso di success=false
                   (errore) */
            }
            @Override
            public void onSuccess(JSONObject data, String message) {
               /* azione da intraprendere in caso di success=true */
            }
         }).execute();
}
```

Listato 4.7: Esempio chiamata ad un servizio

Com'è possibile notare dalla porzione di codice soprastante viene utilizzato un *AsyncTask* per il download dei dati dal server, più precisamente un *NetworkSendAsyncTask*, facente parte sempre della libreria Network.

Questa scelta è stata adottata perché, se si utilizzasse lo stesso thread che si occupa di gestire l'interfaccia grafica, l'utente sarebbe impossibilitato ad interagire con l'applicazione fino al completamento dello scaricamento dei dati.

Di seguito viene riportato il diagramma di sequenza relativo all'operazione di login.

Tale sequenza di operazioni viene effettuata ogni volta che l'applicazione richiede dati al server. Ovviamente il metodo richiamato e i parametri passati cambiano a seconda del servizio invocato.

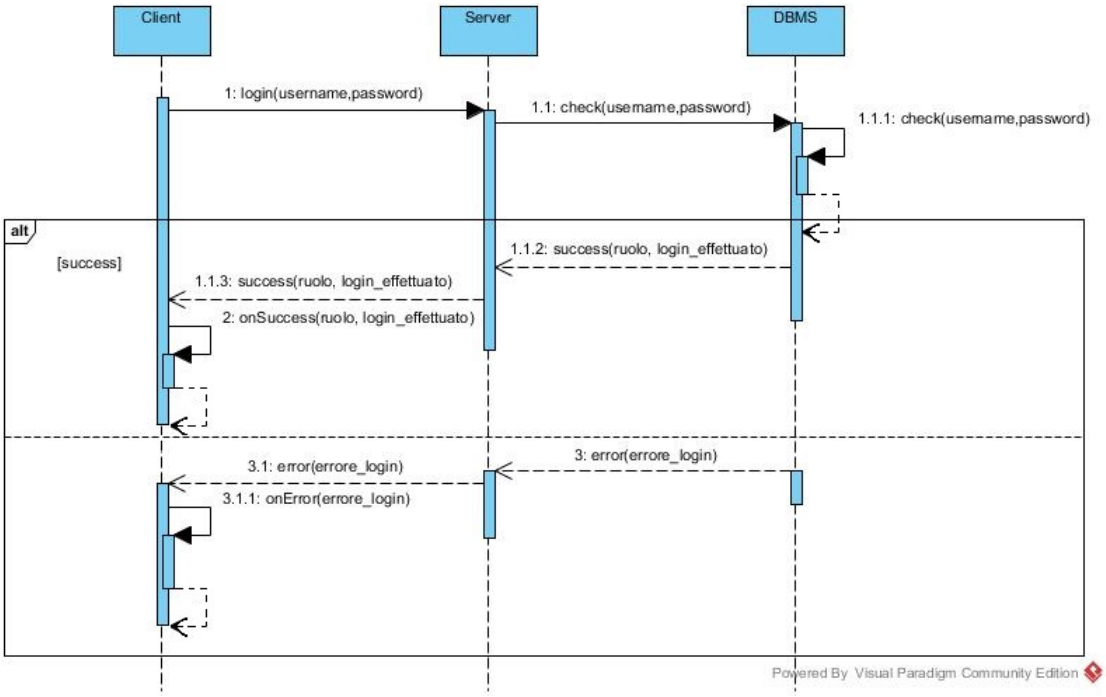


Figura 4.3: Diagramma di sequenza login

4.2.7 Scelte implementative

Sessione applicazione

A livello applicazione l'avvenuto login da parte dell'utente viene registrato nelle SharedPreferences.

All'apertura dell'applicazione viene effettuato un controllo per verificare se la SharedPreferences è settata o meno. Se non è settata verrà mostrata la schermata di login. Se è settata significa che un utente è loggato e l'applicazione si aprirà sulla schermata principale a seconda del ruolo dell'utente, saltando la schermata iniziale di login.

Quando l'utente effettua il logout dall'applicazione, la SharedPreferences relativa al login viene svuotata e l'utente viene reindirizzato alla schermata iniziale per poter effettuare un nuovo accesso.

Utilizzo degli Adapter e dei ViewHolder

Le liste di progetti/ordini/attività, di ticket o di allegati, di messaggi relativi ad un ticket sono presentate all'interno dell'applicazione, facendo uso di Adapter e ListView o GridView.

L'Adapter permette l'accesso ai dati e crea la View corrispondente, sulla base del layout fornito, per ogni elemento del dataset.

All'interno dell'Adapter è stato utilizzato il **ViewHolder pattern** per limitare il numero di chiamate al findViewById(). In questo modo tale metodo viene richiamato una volta sola per tutti gli elementi della lista passata come parametro nel costruttore all'Adapter ed i riferimenti agli elementi del layout vengono memorizzati in una istanza del ViewHolder, associato alla convertView grazie a View.setTag(), per essere utilizzati per gli elementi successivi.

Senza l'utilizzo di questo pattern findViewById() verrebbe richiamato per ogni elemento del layout di ogni elemento della lista, andando a diminuire le velocità con cui la ListView/GridView renderizza i dati.

Segue un esempio di quanto appena detto relativo a FragmentTicket, che mostra la lista dei ticket dell'utente. Dall'esempio sono stati omessi i metodi e i campi di FragmentTicket e TicketAdapter non necessari a comprendere il funzionamento di quanto appena detto.

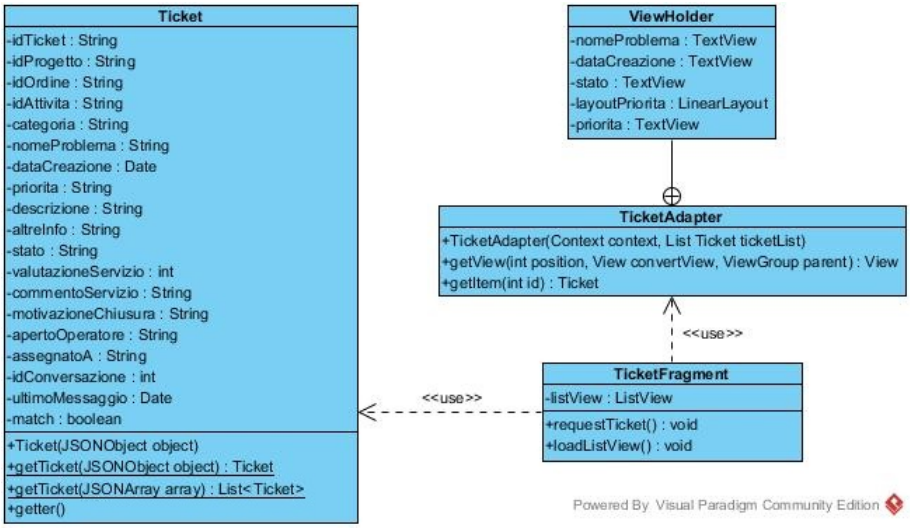


Figura 4.4: Diagramma delle classi Adapter ViewHolder

Push notification

Una push notification è un messaggio che viene inviato dal server al client, in modo asincrono, nel momento in cui vengono resi disponibili nuovi dati, senza che il client debba farne richiesta.

Per l'implementazione delle push notification in Android è stato utilizzato un servizio chiamato **CGM (Google Cloud Messaging)**, integrato attraverso l'inclusione dei Google Play Services nel Gradle di app. Tramite l'utilizzo di GCM l'applicazione invia le notifiche alla piattaforma GCM che le inoltra all'applicazione destinataria.

Di seguito viene mostrato il funzionamento del servizio [11].

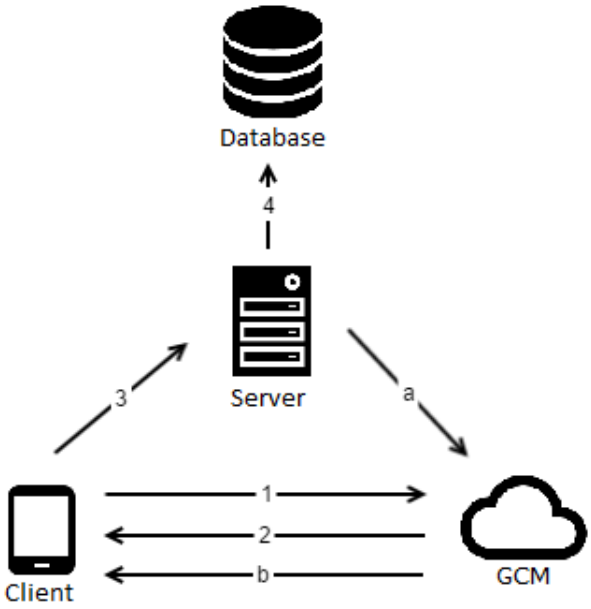


Figura 4.5: Funzionamento GCM

1. il dispositivo invia il *sender id* al server GCM per la registrazione
2. se la registrazione va a buon fine il GCM server risponde con il *registration id*
3. il registration id ricevuto viene inviato al web server
4. il web server salva il registration id sul database per usi futuri

a. quando c'è bisogno di una push notification il server web invia un messaggio al GCM server insieme al registration id recuperato dal database

b. il GCM server recapita il messaggio al dispositivo giusto utilizzando il registration id

Prima di implementare tale meccanismo si è dovuto

- registrare il progetto presso Google Developer Console per ottenere il *sender id* (id assegnato al progetto) che viene mantenuto all'interno dell'app Android,
- registrare una chiave di autenticazione (*Server Key*) utilizzata per ogni richiesta inoltrata a GCM che viene mantenuta sul server web.

All'interno del modulo app, nel package **gcm** sono contenute le classi che si occupano di invocare i metodi per registrarsi al GCM server e gestire i Google Cloud Message.

Quando un utente effettua il login all'applicazione viene richiamato un metodo (*getGcmRegId(activity, listener)*) che si occupa di recuperare il registration id, se presente all'interno delle SharedPreferences perché l'utente aveva già effettuato il login, oppure di inviare il sender id al server GCM per la registrazione.

Il server GCM, in caso di successo, risponde con un registration id che viene salvato all'interno delle SharedPreferences ed invoca un metodo di ServerUtils per il salvataggio di questo sul database (*registraUtenteGCM(username, regId)*).

Quando il server GCM inoltra un messaggio all'applicazione il messaggio viene intercettato da un BroadcastReceiver che si occupa di indicare il Service che si andrà ad occupare dell'intent. All'interno di questo Service sono presenti i metodi per la creazione di diversi tipi di notifiche a seconda dei dati passati come oggetti json dal server.

Le operazioni che danno luogo ad una notifica sono le seguenti:

- ricezione di un nuovo messaggio all'interno di una conversazione legata ad un ticket,
- passaggio di un ticket in gestione,
- aggiunta di un nuovo ticket,
- aggiunta di un nuovo progetto/ordine/attivita.

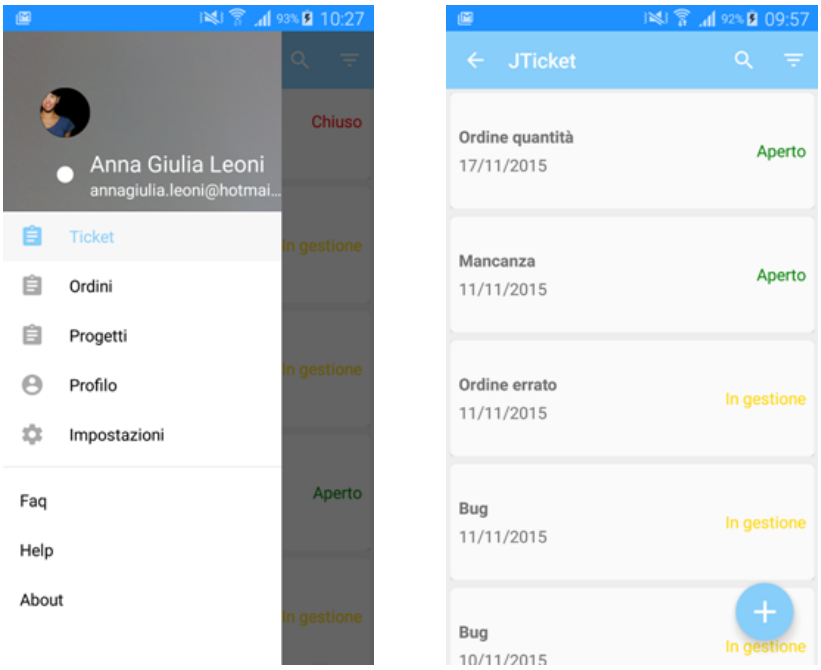
Per il momento, alla ricezione di una notifica, non vengono mai aggiornate le Activity sullo stack se l'applicazione è in esecuzione ed, al click sulla notifica, viene semplicemente aperta l'applicazione e non l'Activity appropriata.

4.2.8 Screenshot schermate principali

Di seguito vengono riportati gli screenshot e le funzionalità contenute all'interno delle schermate principali dell'applicazione sviluppata.

Gli screenshot sottostanti rappresentano il pannello di navigazione ed il Fragment che viene visualizzato sull'Activity principale relativa al cliente.

I pulsanti contenuti nella Toolbar del Fragment in questione danno all'utente la possibilità di ordinare i ticket sulla base della data di creazione e di ultimo messaggio ricevuto oppure di ricercare i ticket sulla base delle parole inserite all'interno della SearchView che si apre cliccando sulla lente di ingrandimento.



(a) Screenshot pannello di navigazione

(b) Screenshot TicketFragment

Figura 4.6: Screenshot schermata principale

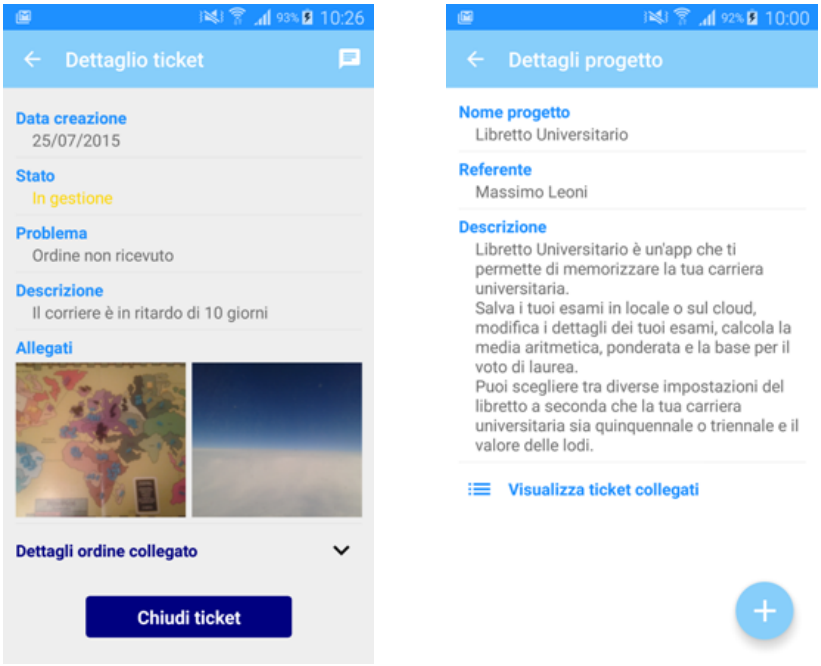
Al click su uno dei ticket contenuti nella ListView si viene indirizzati alla schermata di dettaglio del ticket. La stessa cosa accade quando l'utente clicca su un progetto/ordine/attività rispettivamente da ProgettoFragment/OrdineFragment/AttivitaFragment.

Il dettaglio del ticket contiene le informazioni relative al progetto/ordine/attività a cui è collegato. Se lo stato fosse "Aperto" e fosse loggato un utente con ruolo di Manager sarebbe visibile un pulsante identico a "Chiudi ticket" per effettuare l'assegnazione.

Per la rappresentazione degli allegati è stata scelta una GridView.

Dal dettaglio del progetto è possibile aggiungere direttamente un ticket relativo al progetto saltando la selezione iniziale ed è possibile visualizzare tutti i ticket collegati a questo.

Le immagini seguenti rappresentano quanto appena detto.



(a) Screenshot dettaglio ticket

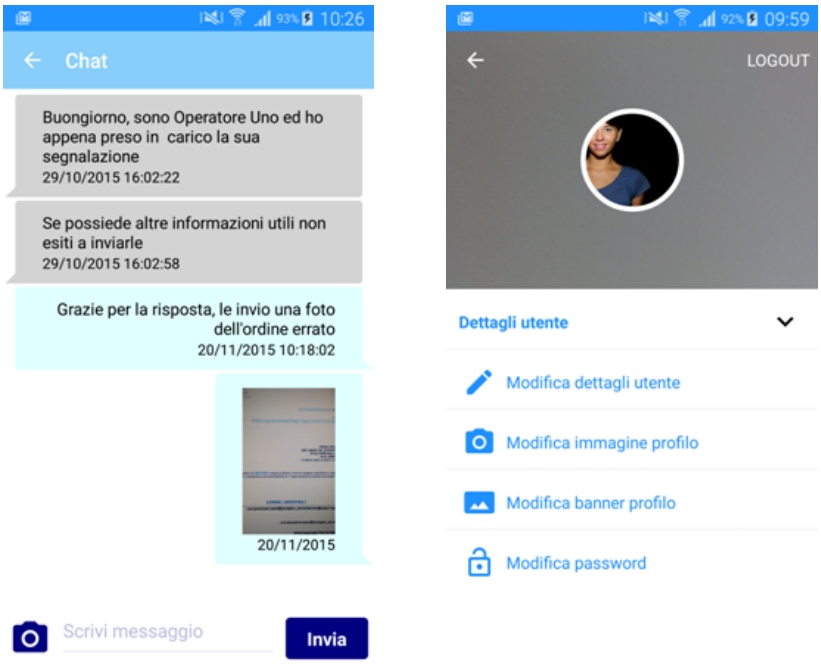
(b) Screenshot dettaglio progetto

Figura 4.7: Screenshot Activity dettagli

Infine gli screenshot relativi alla conversazione associata al ticket e al profilo dell'utente.

Ogni messaggio visualizzato all'interno della ListView che racchiude la conversazione legata al ticket può avere due tipi di layout. La scelta di quale layout utilizzare per ogni messaggio della lista avviene all'interno dell'Adapter tramite un confronto tra l'utente loggato e l'autore del messaggio: se equivalgono verrà scelto il layout con gravità a destra e sfondo azzurro, viceversa altrimenti.

Il profilo dell'utente è stato realizzato esattamente come previsto in fase di progettazione.



(a) Screenshot conversazione (b) Screenshot profilo utente

Figura 4.8: Screenshot Chat e Profilo

Capitolo 5

Testing

I test sull'applicazione sono stati effettuati utilizzando i dispositivi fisici forniti dall'azienda. In questo modo è stato possibile utilizzare la fotocamera senza la necessità di emularla ed i Google Play Services. L'utilizzo dell'emulatore sarebbe stato molto costoso in termini di memoria e performace.

I dispositivi utilizzati sono

- Samsung Galaxy S4, versione Android 5.0.1 (Lollipop - API 21) per il test su smartphone,
- Tablet Lenovo A7600-H android 4.4.2 (KitKat - API 19) per il test su tablet.

I layout dell'applicazione sono stati dimensionati in modo da essere proporzionati alle dimensioni dello schermo del dispositivo che sta eseguendo l'applicazione. In questo modo l'interfaccia grafica risulta piacevole e proporzionata su schermi di dimensione diversa.

5.1 Latenza

Di seguito vengono riportati alcuni dati relativi all'intervallo di tempo che intercorre tra richiesta e risposta dei ticket lato applicazione (chiamata al servizio *getTicket(username)*), a diversi livelli di riempimento del database.

Il tempo di richiesta viene preso in corrispondenza di onPreExecute, all'interno dell'AsyncTask, quello di risposta in corrispondenza di onPostExectute.

Intervallo medio di richiesta-risposta relativo allo scaricamento di ticket

- 10 ticket = 0.426 sec,
- 1000 ticket = 2.140 sec,
- 2000 ticket = 4.121 sec.

Conclusioni

Il capitolo sul testing conclude la trattazione del sistema software esaminato nella presente tesi.

Questa esperienza di tesi è stata molto formativa perché mi ha permesso di lavorare all'interno di un'azienda, prendendo parte al contesto lavorativo, e mettere in pratica ed ampliare le conoscenze accademiche acquisite durante il mio percorso di studi.

L'applicazione è stata sviluppato secondo i requisiti richiesti dall'azienda, che ha tenuto conto delle mie idee e mi ha lasciato libertà in tutte le fasi di sviluppo.

Sviluppi Futuri

L'applicazione sviluppata è già utilizzabile, ma si presta all'aggiunta di altre funzionalità e all'integrazione con un altro software progettato e sviluppato da un collega tesista, Matteo.

Elenco ora di seguito alcune funzionalità che si potrebbero aggiungere o modificare per rendere il software più completo, ma non escludo possano essercene anche delle altre

- permettere solo all'operatore ed ai manager di chiudere i ticket da loro creati: per il momento, infatti, all'operatore ed ai manager non è concesso chiudere alcun tipo di ticket perché, per come è implementato il software, la chiusura del ticket richiede anche di fornire una valutazione sul servizio di assistenza offerto, e non avrebbe senso che loro fornissero una valutazione sul proprio lavoro perché sarebbe non obiettivo,

- permettere all'utente di visualizzare solo i ticket da lui aperti e non anche quelli aperti dagli operatori e dai manager: se l'operatore si accorge, ad esempio, di un bug in un software di un progetto consegnato al cliente, può aprire un ticket per segnalare il problema e non ha senso che anche l'utente visualizzi questo ticket per un problema che lui non aveva notato nel software,

- alla ricezione di una notifica aggiornare le Activity sullo stack se necessario con le informazioni giunte insieme alla notifica ed, al click sulla notifica, aprire l'Activity appropriata

Spiego ora brevemente in cosa consiste il software sviluppato da Matteo e come i due si potrebbero integrare.

Il sistema software sviluppato da Matteo segue le fasi del progetto, dalla discussione con cliente delle specifiche del software richiesto e la redazione del preventivo, fino allo sviluppo del progetto, la divisione in moduli e la loro assegnazione agli sviluppatori dell'azienda nel caso il cliente accetti il preventivo.

Tale software è stato progettato e sviluppato separatamente da quello trattato in questa tesi e l'integrazione con tale software, che utilizza le stesse tecnologie Client e Server utilizzate all'interno di questo progetto, potrebbe avvenire tramite l'unione dei due database sulla tabella Utente, utilizzata per l'autenticazione all'applicazione, e la tabella Progetto.

Ringraziamenti

Desidero ringraziare tutti quelli che, a loro modo, mi hanno supportato nella stesura della tesi e nella realizzazione del progetto.

Ringrazio in primo luogo il Dott. Mirko Ravaioli e l'azienda Appalla per avermi dato la possibilità di sviluppare il progetto presso la loro struttura e per la disponibilità e l'assistenza datami durante tutto il periodo di sviluppo del progetto.

Ringrazio i miei amici, per avermi accompagnato in questi anni universitari, senza i quali alcune lezioni sarebbero state interminabili.

Ringrazio tutta la mia famiglia per avere sempre creduto nelle mie capacità.

Un ringraziamento speciale va ai miei genitori, Massimo e Stefania, per il loro sostegno morale: li ringrazio perché mi hanno sempre spronato e dato la forza per arrivare a questo importante traguardo.

Elenco delle figure

1.1 Schema interfacciamento applicazione e database 2

2.1 Diagramma dei casi d'uso 8

3.1 Schema ER 15
3.2 Entità figlie nel caso di collasso verso il basso 17
3.3 Gerarchia dopo il collasso verso l'alto 17
3.4 Schema logico 20
3.5 Mockup schermata login 24
3.6 Mockup pannello di navigazione Cliente 25
3.7 Mockup apertura di un ticket 26
3.8 Mockup gestione di un ticket 27
3.9 Mockup gestione del profilo 28

4.1 Diffusione dei principali sistemi operativi mobile Ottobre 2015 . 36
4.2 UML classe Progetto 39
4.3 Diagramma di sequenza login 43
4.4 Diagramma delle classi Adapter ViewHolder 44
4.5 Funzionamento GCM 45
4.6 Screenshot schermata principale 47
4.7 Screenshot Activity dettagli 48
4.8 Screenshot Chat e Profilo 49

Bibliografia

[1] Francesco Destri, *Google: gli smartphone superano i PC nelle ricerche online*, http://www.cwi.it/google-gli-smartphone-superano-i-pc-nelle-ricerche-online_80583/, 13 Ottobre 2015

[2] Dario Maio, *Dispense di Basi di Dati*, Funzionalità DBMS p15, Anno Scolastico 2013-2014

[3] *Introduzione a JSON*, http://www.json.org/index.html, 10 Novembre 2015

[4] Danilo Petrozzi, *Funzioni hash: a cosa servono e perché dovresti conoscerle*, http://www.danilopetrozzi.it/funzioni-hash-a-cosa-servono-e-perche-dovresti-conoscerle-2001705, 12 novembre 2015

[5] Evoluzioni Web Società Cooperativa, *App: Native, Ibride o Web?*, http://www.evoluzioniweb.it/IT/Mobile_App_Native_Ibride_Web, 13 novembre 2015

[6] Net Market Share, *Market Share Statistics for Internet Technologies*, https://www.netmarketshare.com/, 13 Novembre 2015

[7] Wikipedia, *Android*, //it.wikipedia.org/w/index.php?title=Android&oldid=76738587, 21 Novembre 2015 19:10 UTC

[8] Wikipedia, *iOS*, //it.wikipedia.org/w/index.php?title=IOS&oldid=76731454, 21 Novembre 2015 11:16 UTC

[9] Wikipedia, *Windows Phone*, //it.wikipedia.org/w/index.php?title=Windows_Phone&oldid=76290675, 7 novembre 2015 22:34 UTC

[10] Android Open Source Project, *Android Studio*, https://developer.android.com/tools/studio/index.html, 18 Novembre 2015

[11] Google Developer, *Cloud Messaging*, `https://developers.google.com/cloud-messaging/gcm`, 18 Novembre 2015