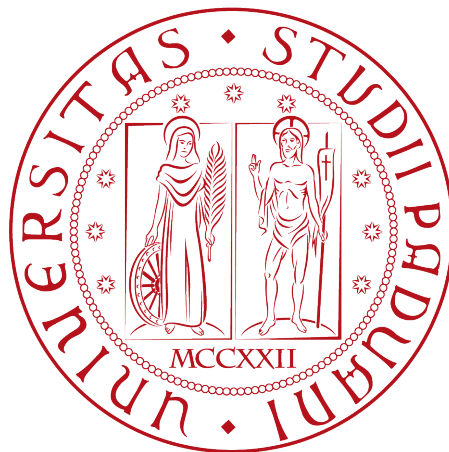


Università degli Studi di Padova

Dipartimento di matematica  
Corso di laurea in informatica



# Gestione e controllo di impianti d'allarme tramite applicazione mobile

Tesi di laurea triennale

Anno accademico 2015/2016

Laureando: Matteo Bortolazzo  
Relatore: Gilberto Filè

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Descrizione del progetto . . . . .	1
1.1.1	Integrazione con l'ecosistema esistente . . . . .	1
1.1.2	Obiettivi . . . . .	1
1.1.3	Gestione degli impianti . . . . .	2
1.2	Interesse aziendale . . . . .	2
1.3	Obiettivi personali . . . . .	3
<b>2</b>	<b>Lo stage</b>	<b>4</b>
2.1	Obiettivi . . . . .	4
2.2	Strumenti . . . . .	4
2.2.1	Visual Studio 2015 . . . . .	4
2.2.2	Visual Studio Team Services (VTS) . . . . .	5
2.2.3	Telegram . . . . .	6
2.2.4	Emulatori multiplatforma . . . . .	6
2.3	Tecnologie . . . . .	7
2.3.1	Xamarin.Forms . . . . .	7
2.3.1.1	Vantaggi . . . . .	8
2.3.1.2	Svantaggi . . . . .	9
2.3.2	MVVM (Model View ViewModel) . . . . .	9
2.3.3	Google Cloud Messaging (GCM) . . . . .	9
2.3.4	RESTful API HTTP . . . . .	10
2.3.5	Portable Class Library (PCL) . . . . .	10
2.4	Metodi di lavoro . . . . .	10
2.4.1	Movimento <i>agile</i> . . . . .	10
2.4.2	<i>Scrum</i> . . . . .	11
2.4.3	Applicazione metodologia . . . . .	12
<b>3</b>	<b>Lavoro svolto</b>	<b>13</b>
3.1	Piano di lavoro . . . . .	13
3.1.1	Pianificazione del lavoro . . . . .	13
3.1.2	Stesura del piano di lavoro . . . . .	14
3.1.3	Requisiti . . . . .	16
3.2	Analisi dei requisiti . . . . .	18
3.2.1	Tecniche utilizzate . . . . .	18
3.2.2	Casi d'uso . . . . .	19
3.2.2.1	UC1 - Accesso . . . . .	19
3.2.2.2	UC1.1 - Esecuzione accesso . . . . .	20
3.2.2.3	UC1.1.1 - Inserimento username . . . . .	20
3.2.2.4	UC1.1.2 - Inserimento password . . . . .	20
3.2.2.5	UC1.1.3 - Selezione server . . . . .	21
3.2.2.6	UC1.2 - Visualizzazione errore credenziali errate . . . . .	21
3.2.2.7	UC1.3 - Visualizzazione errore mancanza internet . . . . .	21
3.2.2.8	UC2 - Visualizzazione schermata principale . . . . .	21

3.2.2.9	UC2.1 - Visualizzazione lista impianti . . . . .	22
3.2.2.10	UC2.2 - Visualizzazione dettagli impianto . . . . .	22
3.2.2.11	UC2.2.1 - Visualizzazione storico eventi . . . . .	23
3.2.2.12	UC2.2.2 - Visualizzazione lista sensori . . . . .	23
3.2.2.13	UC2.2.3 - Visualizzazione lista gruppi . . . . .	24
3.2.2.14	UC2.2.4 - Visualizzazione lista utenti impianto . . . . .	24
3.2.2.15	UC2.2.5 - Visualizzazione lista comandi impianto . . . . .	24
3.2.2.16	UC2.2.6 - Visualizzazione lista <i>ticket</i> impianto . . . . .	24
3.2.2.17	UC2.2.7 - Visualizzazione storico interventi . . . . .	25
3.2.2.18	UC2.2.8 - Visualizzazione dettaglio evento . . . . .	25
3.2.2.19	UC2.2.8.1 - Visualizzazione informazioni avanzate evento . . . . .	25
3.2.2.20	UC2.2.8.2 - Visualizzazione eventuali immagini dell'evento . . . . .	26
3.2.2.21	UC2.2.8.3 - Apertura <i>ticket</i> per evento . . . . .	26
3.2.2.22	UC2.2.9 - Esecuzione comando . . . . .	26
3.2.2.23	UC2.2.10 - Creazione nuovo <i>ticket</i> per impianto . . . . .	26
3.2.2.24	UC2.2.10.1 - Inserimento nome di riferimento . . . . .	27
3.2.2.25	UC2.2.10.2 - Inserimento titolo <i>ticket</i> . . . . .	27
3.2.2.26	UC2.2.10.3 - Inserimento descrizione . . . . .	27
3.2.2.27	UC2.2.11 - Visualizzazione dettagli specifici . . . . .	28
3.2.2.28	UC2.2.12 - Visualizzazione contatti impianto . . . . .	28
3.2.2.29	UC2.2.13 - Selezione contatto . . . . .	28
3.2.2.30	UC2.2.13.1 - Chiamata a contatto . . . . .	29
3.2.2.31	UC2.2.13.2 - Nuova <i>email</i> a contatto . . . . .	29
3.2.2.32	UC2.2.14 - Visualizzazione errore mancanza internet . . . . .	29
3.2.2.33	UC2.3 - Visualizzazione impostazioni . . . . .	29
3.2.2.34	UC2.4 - Visualizzazione lista <i>ticket</i> . . . . .	29
3.2.2.35	UC2.5 - Visualizzazione dettaglio <i>ticket</i> . . . . .	30
3.2.2.36	UC2.5.1 - Visualizzazione informazioni avanzate . . . . .	30
3.2.2.37	UC2.5.2 - Visualizzazione <i>chat ticket</i> . . . . .	30
3.2.2.38	UC2.5.3 - Invio messaggio in <i>chat</i> . . . . .	30
3.2.2.39	UC2.6 - Visualizzazione interventi assegnati . . . . .	31
3.2.2.40	UC2.7 - Visualizzazione dettaglio intervento . . . . .	31
3.2.2.41	UC2.7.1 - Visualizzazione informazioni avanzate . . . . .	31
3.2.2.42	UC2.7.2 - Visualizzazione informazioni impianto . . . . .	32
3.2.2.43	UC2.7.3 - Presa in carico intervento . . . . .	32
3.2.2.44	UC2.7.4 - Apertura intervento . . . . .	32
3.2.2.45	UC2.7.5 - Chiusura intervento . . . . .	32
3.2.2.46	UC2.7.6 - Sospensione intervento . . . . .	32
3.2.2.47	UC2.7.7 - Visualizzazione errore mancanza internet . . . . .	33
3.2.2.48	UC2.8 - Visualizzazione errore mancanza internet . . . . .	33
3.2.3	Suddivisione dei requisiti . . . . .	33
3.2.3.1	Requisiti funzionali . . . . .	33

3.2.3.2	Requisiti di vincolo . . . . .	34
3.2.3.3	Requisiti di qualità . . . . .	35
3.2.3.4	Requisiti prestazionali . . . . .	35
3.2.4	Considerazioni sui requisiti . . . . .	36
3.3	Vincoli di progetto . . . . .	36
3.3.1	Dal progetto . . . . .	36
3.3.2	Dai processi aziendali . . . . .	36
3.4	Studio delle tecnologie . . . . .	37
3.4.1	Linguaggi . . . . .	37
3.4.2	Progetti nativi . . . . .	37
3.4.3	Ambienti di sviluppo . . . . .	38
3.4.4	Strumenti di <i>team</i> . . . . .	38
3.4.5	<i>Framework</i> e librerie . . . . .	39
3.5	Sviluppo . . . . .	40
3.5.1	Struttura dei progetti . . . . .	40
3.5.2	Funzioni base . . . . .	41
3.5.2.1	Richieste alle <i>API web</i> . . . . .	41
3.5.2.2	Scrittura su disco . . . . .	42
3.5.2.3	<i>Design</i> . . . . .	42
3.5.2.4	Lista impianti . . . . .	43
3.5.2.5	Dettagli impianto . . . . .	44
3.5.2.6	Comandi . . . . .	45
3.5.2.7	Creazione <i>ticket</i> . . . . .	46
3.5.2.8	Impostazioni . . . . .	47
3.5.3	Funzioni avanzate . . . . .	48
3.5.3.1	Chat <i>ticket</i> . . . . .	48
3.5.3.2	Gestione interventi . . . . .	49
3.5.3.3	Geolocalizzazione . . . . .	50
3.5.3.4	Chiamate e invio <i>email</i> . . . . .	51
3.5.3.5	Notifiche <i>push</i> . . . . .	51
4	<b>Testing</b> . . . . .	<b>52</b>
4.1	Analisi statica . . . . .	52
4.2	Analisi dinamica . . . . .	52
5	<b>Conclusione</b> . . . . .	<b>54</b>
5.1	Obiettivi raggiunti . . . . .	54
5.1.1	Imposti dall'azienda . . . . .	54
5.1.2	Propri del progetto . . . . .	55
5.1.3	Personalizzati . . . . .	55
5.2	Difficoltà affrontate . . . . .	55
5.3	Bilancio formativo . . . . .	56

## Elenco delle figure

1	Logo ufficiale della <i>suite</i> PointSecurity Service (PSS). . . . .	1
2	Logo ufficiale dell'applicazione PointSecurity Mobile. . . . .	2
3	Logo di <i>Visual Studio 2015</i> . . . . .	5
4	Logo di Visual Studio Team Services. . . . .	5
5	Applicazione Xamarin.Forms eseguita sui diversi emulatori. . . .	7
6	Logo ufficiale di Xamarin. . . . .	8
7	Applicazione ufficiale di esempio per Xamarin.Forms eseguita sui tre sistemi operativi. . . . .	8
8	Struttura base del pattern <i>MVVM</i> . . . . .	9
9	Schema base della comunicazione tra l' <i>API web</i> e i <i>client</i> . . . . .	10
10	Schema del flusso di lavoro base dell'ideologia <i>agile</i> . . . . .	11
11	Flusso di lavoro Scrum. . . . .	12
12	Tabella riassuntiva con la pianificazione delle macro attività. . .	15
13	Diagramma di Gantt del piano di stage. . . . .	16
14	Grafico a barre rappresentante i requisiti suddivisi per tipologia.	17
15	Diagramma <i>UML</i> del caso d'uso UC1 - Accesso. . . . .	19
16	Diagramma <i>UML</i> del caso d'uso UC1.1 - Esecuzione accesso. . .	20
17	Diagramma <i>UML</i> del caso d'uso UC2 - Visualizzazione schermata principale. . . . .	22
18	Diagramma <i>UML</i> del caso d'uso UC2.2 - Visualizzazione dettagli impianto. . . . .	23
19	Diagramma <i>UML</i> del caso d'uso UC2.8 - Visualizzazione dettaglio storico. . . . .	25
20	Diagramma <i>UML</i> del caso d'uso UC2.2.10 - Creazione nuovo <i>ticket</i> per impianto. . . . .	27
21	Diagramma <i>UML</i> del caso d'uso UC2.2.13 - Selezione contatto. .	28
22	Diagramma <i>UML</i> del caso d'uso UC2.5 - Visualizzazione dettaglio <i>ticket</i> . . . . .	30
23	Diagramma <i>UML</i> del caso d'uso UC2.7 - Visualizzazione dettaglio intervento. . . . .	31
24	Logo di Telegram. . . . .	37
25	Finestra di esempio di Visual Studio. . . . .	38
26	Finestra di esempio del portale <i>web</i> Visual Studio Team Services per la gestione di un progetto Scrum. . . . .	39
27	Copertina del libro ufficiale per lo studio di <i>Xamarin.Forms</i> . . .	40
28	Progetti generati da Visual Studio. . . . .	40
29	Cartella dei tre progetti Xamarin.Forms finiti. . . . .	41
30	<i>Screenshot</i> del menu principale (Android). . . . .	43
31	<i>Screenshot</i> della pagina contenente la lista di impianti visualiz- zabili da un tecnico (iOS). . . . .	44
32	<i>Screenshot</i> della pagina di dettaglio dell'impianto (Android). . .	45
33	<i>Screenshot</i> della tab per comandare l'impianto (Windows). . . .	46
34	<i>Screenshot</i> della pagina di creazione di un nuovo <i>ticket</i> (Windows).	47
35	<i>Screenshot</i> della pagina delle impostazioni (iOS). . . . .	48

36	<i>Screenshot</i> della <i>chat</i> di un <i>ticket</i> (Android). . . . .	49
37	<i>Screenshot</i> della pagina per la gestione di un intervento da parte di un tecnico (Android). . . . .	50
38	Schema di esempio per le notifiche <i>push</i> di iOS. . . . .	51
39	Lista degli emulatori disponibili durante lo sviluppo. . . . .	52
40	Grafico copertura requisiti. . . . .	55

## Elenco delle tabelle

1	Tabella dei requisiti funzionali scritti con il <i>tutor</i> durante l'analisi dei requisiti . . . . .	34
2	Tabella dei requisiti di vincolo scritti con il <i>tutor</i> durante l'analisi dei requisiti . . . . .	35
3	Tabella dei requisiti di qualità scritti con il <i>tutor</i> durante l'analisi dei requisiti . . . . .	35
4	Tabella dei requisiti prestazionali scritti con il <i>tutor</i> durante l'analisi dei requisiti . . . . .	35

## Convenzioni tipografiche

Durante la stesura del documento adottato le seguenti norme tipografiche:

- Utilizzo il *corsivo* per le parole in lingua straniera.
- Indico con una <sub>G</sub> tutti i termini che necessitano di una spiegazione esplicita, presente nel glossario a fine documento.

# 1 Introduzione

## 1.1 Descrizione del progetto

L'obiettivo dello *stage* è il rinnovo completo dell'applicazione *mobile* ufficiale a supporto della *suite* di *software* PointSecurity Service. I vari *software* della *suite* si occupano principalmente della centralizzazione e gestione di allarmi provenienti dagli impianti di sicurezza installati in varie sedi, dai privati alle banche. La struttura di PointSecurity Service è modulare, esistono quattro programmi principali ed indispensabili: un ricettore (modulare a sua volta), un elaboratore, un gestionale ed un visualizzatore. Ogni altro modulo (o programma) aggiunge nuove funzioni ed è acquistabile separatamente. Inoltre, in parallelo alla realizzazione della nuova applicazione, è iniziato lo sviluppo di un sistema di *ticketing* per le problematiche legate agli impianti e all'ottimizzazione dei processi gestionali interni.

### 1.1.1 Integrazione con l'ecosistema esistente

Per permettere ai clienti di gestire i propri impianti da remoto esiste una piattaforma *web*, sincronizzata ma non direttamente collegata con il *mainframe*, per garantire la sicurezza e la stabilità di quest'ultimo. Il *server web* espone delle *API* che permettono alle applicazioni di interagire con esso, in particolare viene interrogato direttamente in caso di lettura e viene utilizzato come *proxy* verso il *mainframe* in scrittura. Ogni richiesta *web* viene inviata con protocollo *HTTPS* e criptata con chiave asimmetrica a 256bit per garantire il massimo della sicurezza.



Figura 1: Logo ufficiale della *suite* PointSecurity Service (PSS).

### 1.1.2 Obiettivi

La nuova applicazione vuole migliorare le funzioni già disponibili in quella attuale, l'aspetto grafico e le prestazioni. Inoltre, l'aggiunta di nuove funzioni, come l'integrazione con il nuovo sistema di *ticketing* e la possibilità per il personale autorizzato di gestire gli impianti da remoto, offriranno valore aggiunto.



L'azienda ha richiesto l'utilizzo di Xamarin.Forms 2, una serie di strumenti e librerie che permettono la condivisione massima del codice tra iOS, Android e Windows 10. I linguaggi utilizzati da Xamarin.Forms sono C# e XAML, entrambi sviluppati da Microsoft, uno come linguaggio di programmazione ad alto livello e uno come linguaggio di *markup*. Ho utilizzato un *server* Visual Studio Team Services per la gestione interna del progetto il quale permette ai membri del *team* di condividere codice, tracciare il lavoro e distribuire il *software*.

### 1.1.3 Gestione degli impianti

Per una chiara comprensione dei prossimi capitoli descrivo brevemente come gli impianti d'allarme sono gestiti da PointSecurity Service. Ogni utente ha una serie di impianti a lui associati. Ad ogni impianto sono collegati diversi sensori di vario tipo che fanno scattare differenti tipi di allarme. I sensori possono essere raggruppati per gruppi, questo permette la gestione di più sensori contemporaneamente, ogni sensore può appartenere a più gruppi. Per l'accensione e lo spegnimento degli impianti vengono creati vari utenti in modo d'avere un *log* più preciso delle attività. Quando scatta un allarme o un altro tipo di segnalazione, l'impianto invia al *software* un evento. L'invio può avvenire tramite linea telefonica, tramite IP o GPRS in base al tipo e all'impostazione dell'impianto. Per il controllo remoto alcuni impianti mettono a disposizione dei comandi da inviare tramite IP.

## 1.2 Interesse aziendale

L'azienda, come detto in precedenza, ha l'obiettivo di aggiornare l'applicazione ufficiale corrente. I motivi di questa decisione nascono dalla necessità di offrire al cliente un prodotto migliore con una manutenibilità maggiore, non raggiungibili con le vecchie tecnologie al momento utilizzate. Tra le migliori generali l'azienda voleva un'interfaccia moderna, utilizzando al meglio le linee guide dei vari sistemi operativi, un aumento delle prestazioni grazie alle nuove librerie e la compatibilità con i nuovi dispositivi tramite l'ultima versione dei rispettivi *SDK*.



Figura 2: Logo ufficiale dell'applicazione PointSecurity Mobile.

Le nuove funzioni, invece, sono mirate a velocizzare le operazioni di gestione degli impianti e ottimizzare il lavoro nelle sedi operative. Questi obiettivi sono raggiungibili attuando diverse modifiche:

- Aumentare la responsabilità degli operatori sul campo, fornendo ad essi le funzioni per la gestione degli impianti fisici direttamente da *smartphone*;
- Geolocalizzare i tecnici fuori sede per una gestione ottimizzata dei percorsi e degli interventi imprevisti;
- Permettere ai clienti di contattare l'azienda responsabile dell'impianto in caso di eventuali guasti in modo veloce e tracciabile grazie ad un sistema di *ticketing*.

Per rendere l'applicazione mantenibile nel tempo, l'azienda ha imposto l'utilizzo del pattern MVVM (*Model View ViewModel*) per la divisione del codice grafico da quello logico. Le librerie di supporto dovevano essere gestite in progetti separati. Grazie a questi accorgimenti è possibile essere certi che in caso di modifica o aggiunta di funzioni il codice sia pienamente modulare e di rendere quindi più semplice lo sviluppo.

### 1.3 Obiettivi personali

Il mondo delle tecnologie *mobile* è in continua espansione, gli applicativi non professionali sono oramai tutti presenti anche su *smartphone*. Per questo motivo ho deciso di svolgere uno *stage* che prevedesse la realizzazione di un prodotto per *smartphone*. Usare inoltre uno strumento per lo sviluppo nativo multiplatforma come Xamarin mi incuriosiva in quanto ancora di nicchia (statisticamente parlando). Lo sviluppo non è comunque possibile al 100% con Xamarin.Forms, questo permette di familiarizzare anche con le *API*, la struttura e linguaggi nativi delle diverse piattaforme. Conoscevo già i linguaggi da dover utilizzare (C# e XAML) in quanto appresi durante il percorso formativo antecedente all'Università, ma l'applicazione di essi in ambito *mobile* sarebbe stata una novità.

Al di là dell'aspetto puramente tecnico, uno dei miei obiettivi era il miglioramento delle capacità di lavoro in *team*. Grazie al progetto di Ingegneria del Software avevo già un'idea abbastanza chiara del lavoro di gruppo, ma in un ambiente professionale sarebbe stato sicuramente diverso. Imparare ad usare gli strumenti per il lavoro cooperativo per me era fondamentale per progetti futuri, sia personali che aziendali, nonostante un buon uso di essi durante il progetto citato poc'anzi. Per questi motivi lo *stage* proposto da PointSecurity Software mi è sembrata la scelta ideale, sia a livello personale che curriculare.

## 2 Lo stage

### 2.1 Obiettivi

Durante la stesura del piano di stage io e il *tutor* aziendale vi abbiamo inserito alcuni requisiti. Questi requisiti non sono specifici per il progetto da me svolto, per il quale ho effettuato io stesso l'analisi, ma rappresentano requisiti che l'azienda si aspetta lo stagista soddisfi, al fine di poter valutare positivamente la conclusione dello *stage*. Fin da subito erano stati suddivisi tra necessari (identificati come: «n»): vincolanti in quanto primari e di diretto impatto sulla valutazione dello stage e desiderabili (identificati come «d»): non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiuntivo. I requisiti presenti nel piano di stage sono i seguenti:

- «n» Sviluppo delle *view* per funzionalità basi dell'applicativo *mobile*;
- «n» Sviluppo funzionalità basi dell'applicativo *mobile*;
- «n» Sviluppo funzionalità per interrogazione *API* ad-hoc e visualizzazione delle risposte ricevute;
- «n» Sviluppo funzionalità per salvare dati attraverso *API* ad-hoc;
- «d» Sviluppo delle *view* per funzionalità supplementari nell'applicativo;
- «d» Sviluppo funzionalità supplementari nell'applicativo *mobile*.

### 2.2 Strumenti

#### 2.2.1 Visual Studio 2015

*Visual Studio* è un ambiente di sviluppo integrato di *Microsoft* per sviluppare programmi per *Windows*, siti e servizi *web*, ma anche applicazioni *mobile*. Supporta diversi linguaggi e tecnologie inoltre, grazie all'apposito *SDK* è stato aggiunto il supporto ad altri linguaggi favorendo così sviluppatori di terze parti. Tra le funzioni principali dell'editor si trovano l'autocompletamento del codice, la ricerca incrementale tramite espressioni regolari e vari tipi di *refactoring* del codice. *Visual Studio* mette a disposizione inoltre un potente *debugger* sia a livello di codice che di macchina, esso supporta sia il codice nativo che quello gestito da una macchina virtuale, come il *CRM*.



Figura 3: Logo di *Visual Studio 2015*.

Altri strumenti come l'*editor* grafico e il *data explorer* per i database sono resi disponibili gratuitamente. Infine l'*IDE* si integra con i principali sistemi di versionamento e di *team management* rendendolo un ambiente di sviluppo completo.

### 2.2.2 Visual Studio Team Services (VTS)

I VSTS sono una serie di servizi creati da Microsoft come strumento complementare all'*IDE* Visual Studio. Questi servizi, rappresentati da una serie di strumenti collaborativi *cloud-powered*, sono compatibili con diversi ambienti di sviluppo e rendono possibile la collaborazione di più *team*, in modo efficace, su progetti diversi tra loro.

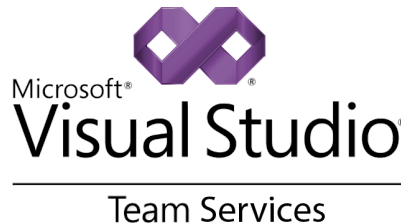


Figura 4: Logo di Visual Studio Team Services.

Di seguito sono presenti le principali caratteristiche e servizi offerti:

- **Compatibilità con diversi linguaggi:** C#, Java, Python, HTML5, JavaScript ed altri;
- **Supporto in diversi IDE:** Visual Studio, Eclipse, IntelliJ ed altri;
- **Controllo codice sorgente:** Illimitate *repository* private usando Git per un controllo di versione distribuito o Team Foundation Version Control (TFVC) per un controllo di versione centralizzato;
- **Strumenti per metodologia agile:** Supporto per Kanban e Scrum collegando gli elementi direttamente al codice;

- **Continuous integration:** Compilazione ed esecuzione dei test ad ogni cambiamento del codice. Abilitabile il *continuous delivery* per pubblicare direttamente i siti *web* che passano i test.
- **Integrazione:** I servizi sono aperti a strumenti di terze parti grazie alle *API REST* aperte. Servizi come GitHub, Trello e Slack sono quindi compatibili;
- **Cloud:** Tutti i servizi sono disponibili in *cloud* grazie alla piattaforma Visual Studio Online.

### 2.2.3 Telegram

Telegram è un *software* multipiattaforma, disponibile su dispositivi *desktop*, *mobile* e sul *web* per la messaggistica istantanea. Tra le funzioni principali del software si sono la possibilità di inviare qualunque tipo di file, rispondere a messaggi specifici e *taggare* le persone. Queste caratteristiche rendono Telegram un sistema di messaggistica molto veloce e comodo per piccoli *team*, è stato infatti adottato per questo progetto per le comunicazioni interne tra sviluppatori e *tester*.

### 2.2.4 Emulatori multipiattaforma

Durante lo stage, per velocizzare lo sviluppo, ho utilizzato gli emulatori delle diverse piattaforme. Gli emulatori per Windows 10 Mobile e Android da me utilizzati sono quelli realizzati da Microsoft. Per quanto riguarda iOS, non essendo presente un emulatore funzionante per il sistema operativo Windows, ho utilizzato un *proxy*, sempre sviluppato da Microsoft, per utilizzare in remoto quello originale di casa Apple, installato su un Mac Mini presente in azienda.

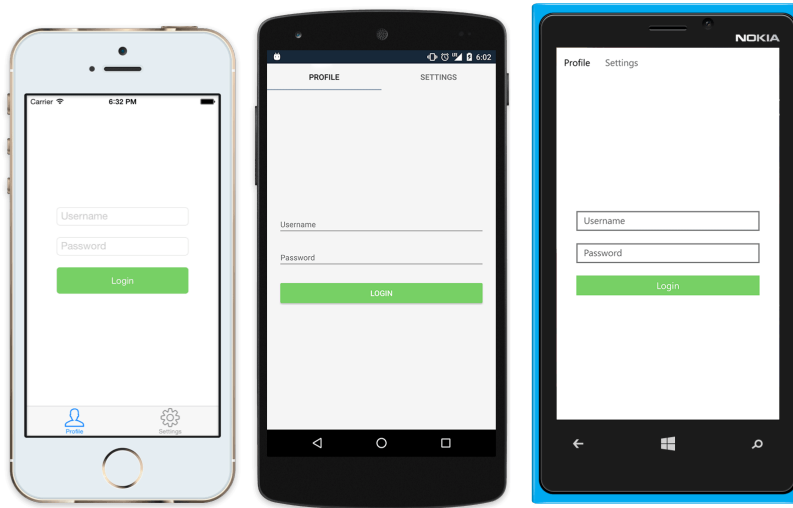


Figura 5: Applicazione Xamarin.Forms eseguita sui diversi emulatori.

Per ogni sistema operativo sono disponibili diversi dispositivi emulabili, che differiscono soprattutto per quantità di RAM, dimensione dello schermo e risoluzione di quest'ultimo. L'emulazione permette di avere il massimo controllo sul dispositivo, dalla velocità di connessione ad *internet*, alla posizione GPS fino allo spazio rimanente su disco. Queste caratteristiche permettono agli emulatori di essere mezzi importantissimi per il *testing* immediato di funzioni e caratteristiche dell'applicazione.

## 2.3 Tecnologie

### 2.3.1 Xamarin.Forms

Xamarin.Forms è un prodotto creato da Xamarin (un'azienda di proprietà Microsoft), costituito da *API* e librerie per lo sviluppo di applicazioni native per iOS, Android e Windows utilizzando lo stesso codice sorgente. I prodotti principali dell'azienda sono Xamarin.iOS e Xamarin.Android i quali permettono lo sviluppo di applicazioni sostituendo il linguaggio originale con il C#. Xamarin.Forms sfrutta i prodotti sopracitati e l'*SDK* di Windows per il proprio funzionamento. Le librerie sono composte da classi rappresentanti elementi comuni a tutte le piattaforme, convertite poi in componenti nativi durante la compilazione.



Figura 6: Logo ufficiale di Xamarin.

### 2.3.1.1 Vantaggi

L'obiettivo di questa tecnologia è permettere agli sviluppatori di utilizzare il numero minimo di risorse per creare applicazioni di qualità. La condivisione quasi totale del codice (96% secondo gli sviluppatori) permette di risparmiare tempo e costi di sviluppo. L'utilizzo ideale risiede negli applicativi per la presentazione di dati e con pochi elementi *platform specific*, perfetta quindi per il progetto di *stage*. La presenza di un unico codice sorgente garantisce un'alta manutenibilità del prodotto, basta infatti un'unica modifica per aggiornare tutte le piattaforme. L'intero progetto è inoltre *open source* permettendo agli sviluppatori di terze parti di creare e modificare librerie e controlli, aumentando così la qualità e la diffusione del progetto.

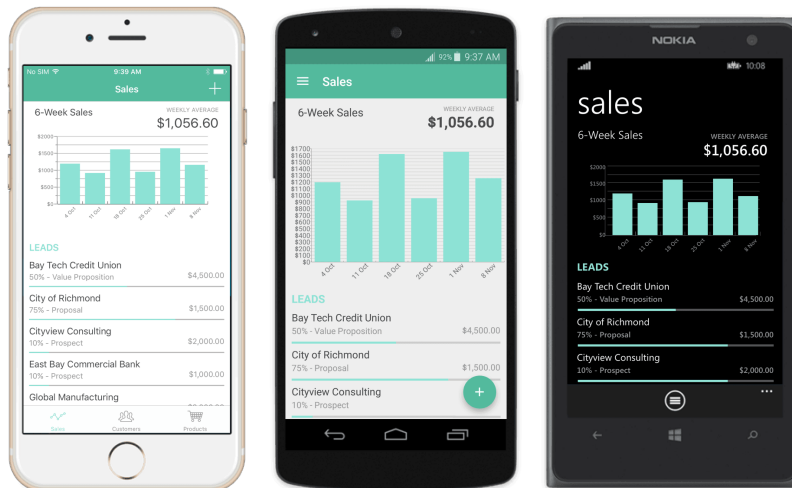


Figura 7: Applicazione ufficiale di esempio per Xamarin.Forms eseguita sui tre sistemi operativi.

### 2.3.1.2 Svantaggi

Gli svantaggi sono eguagliati dai vantaggi: per applicazioni con un alto numero di classi native, Xamarin.Forms è sconsigliato. Modifiche da apportare ad una sola piattaforma, o in modo differente a tutte e 3 possono risultare estremamente difficili: i controlli grafici ne sono un esempio. Può capitare inoltre di riscontrare comportamenti diversi dello stesso codice in base alla piattaforma, il *debugging* può essere difficile per carenza di informazioni specifiche.

### 2.3.2 MVVM (Model View ViewModel)

*Model-view-viewmodel* è un *pattern* architetturale progettato e sviluppato da Microsoft per semplificare lo sviluppo di interfacce grafiche *event-driver* grazie alla separazione tra esse ed il *backend* (o *model*). I *viewModel* hanno la responsabilità di convertire i valori ottenuti interrogando i *model* in elementi grafici direttamente rappresentabili a schermo attraverso le *view*. Nonostante essi siano concettualmente dei *model*, si occupano quasi completamente anche della logica della *view*. Gli elementi più importanti di questo *pattern* sono il concetto di *binding* e di *command*. Il primo è il modo per legare mono/bidirezionalmente un campo presentato nel *viewmodel* con un elemento grafico della *view*. Il secondo sostituisce il concetto di evento, gli eventi non vengono gestiti dalla *view* ma fanno scattare un *command* gestito dal *viewmodel*. Questa tecnologia è utilizzabile nelle tecnologie Microsoft come WPF, Silverlight e Xamarin.Forms.

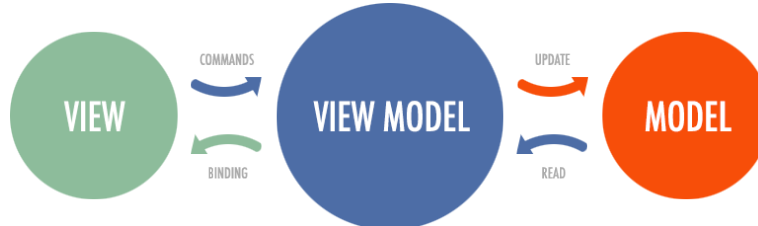


Figura 8: Struttura base del pattern *MVVM*.

### 2.3.3 Google Cloud Messaging (GCM)

Google Cloud Messaging è un servizio offerto da Google per l'invio di informazioni e notifiche dai *server* di terze parti ai dispositivi Android o alle estensioni di Google Chrome. Esso consiste in una serie di *API* e un *SDK*, inoltre l'utilizzo è completamente gratuito. Il GCM ha l'abilità di inviare notifiche *push*, comandi di *deep-linking* e dati. Ogni applicazione che vuole sfruttare il servizio deve registrarsi a esso. Questo servizio genera un codice univoco utilizzabile dagli sviluppatori per comunicare con un dispositivo.



### 2.3.4 RESTful API HTTP

Le *RESTful API HTTP* sono *API* offerte da un servizio *web* che applicano l'architettura *REST* e utilizzano il protocollo *HTTP*. Partendo da un *URL* base, le richieste vengono composte in modo additivo ed inviate utilizzando diversi metodi *HTTP*, ognuno dei quali può rappresentare un'operazione differente. Solitamente le *API* di questo tipo rispondono con oggetti *JSON* o *XML*. Attualmente questo protocollo non prevede degli standard ma solo delle *best practice*, quindi ogni sviluppatore può progettare il proprio servizio in modo differente.

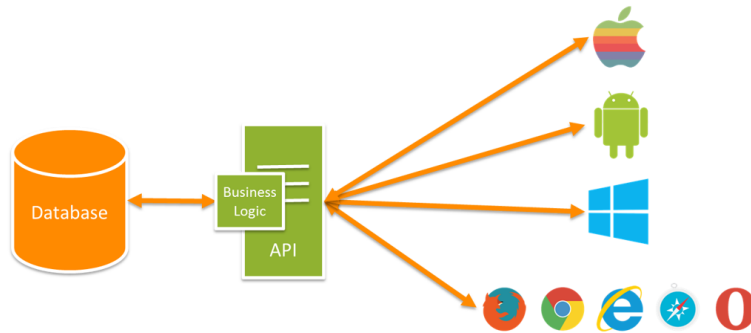


Figura 9: Schema base della comunicazione tra l'*API web* e i *client*.

### 2.3.5 Portable Class Library (PCL)

Le *.NET Framework Portable Class Library* sono un tipo di progetto presente in Visual Studio. Il loro scopo è di aiutare a scrivere applicazioni multipiattaforma in modo semplice e veloce. Le librerie possono avere come *target* molte piattaforme diverse e queste possono essere cambiate in qualsiasi momento modificando le impostazioni di compilazioni. Il prodotto della compilazione è una *Dynamic-link library* (DLL). Le PCL sono un elemento fondamentale nello sviluppo di applicazioni Xamarin.Forms, una libreria compilata correttamente può essere importata in tutti i progetti delle varie piattaforme.

## 2.4 Metodi di lavoro

Il lavoro doveva essere organizzato e gestito utilizzando i servizi messi a disposizione di Visual Studio Team Services seguendo la metodologia *agile* Scrum.

### 2.4.1 Movimento *agile*

L'ideologia *agile* è nata dall'esigenza di rispondere a cambiamenti imprevisti attraverso incrementi, lavoro incrementale a cadenza regolare con *feedback* empirico. I modelli *agile* sono un'alternativa ai modelli a cascata o iterativi usati in passato.

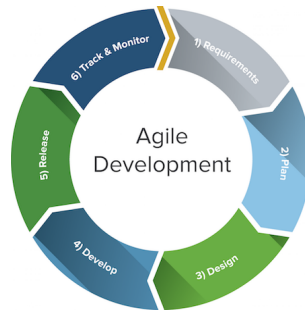


Figura 10: Schema del flusso di lavoro base dell'ideologia *agile*.

### 2.4.2 *Scrum*

*Scrum* è il metodo *agile* più popolare, deve questa popolarità alla sua semplicità e flessibilità. Si basa su *feedback* empirici, sull'autogestione di ogni *team* e sulla volontà di creare un incremento (funzioni o *bug* corretti) testato al prodotto in poche iterazioni. Questa ideologia molto spesso viene applicata male perché si scontra con le abitudini consolidate delle aziende che adottano metodi di sviluppo non *agile*.

*Scrum* prevede tre ruoli: il proprietario del prodotto, il *team* e lo *Scrum master*, il quale ha il dovere di rimuovere eventuali impedimenti durante lo sviluppo. La responsabilità della gestione del progetto viene divisa invece tra i tre ruoli.

I concetti principali di questo metodo sono lo *sprint* e il *backlog*. Il primo è un ciclo di durata temporale variabile (da una settimana ad un mese solitamente) durante il quale si cerca di produrre un incremento significativo al prodotto. Il secondo è una lista prioritaria di elementi da gestire durante lo *sprint*: funzionalità, *bug*, lavoro tecnico e acquisizione di conoscenze.

Infine *Scrum* ha 5 incontri:

- ***Backlog refinement***: Incontro tra il *team* e il proprietario del prodotto per aggiornare il *backlog* per il prossimo *sprint*;
- ***Sprint planning***: Incontro tra tutti e tre i ruoli per definire nello specifico la gestione del lavoro durante lo *sprint*;
- ***Daily Scrum***: Incontro giornaliero tra i membri del *team* di 15 minuti per decidere cosa fare durante la giornata;
- ***Sprint review meeting***: A fine *sprint* illustrano i progressi fatti durante quest'ultimo;
- ***Sprint retrospective meeting***: Ultima cosa da fare a fine *sprint*, utile per analizzare dove si può migliorare a livello lavorativo.

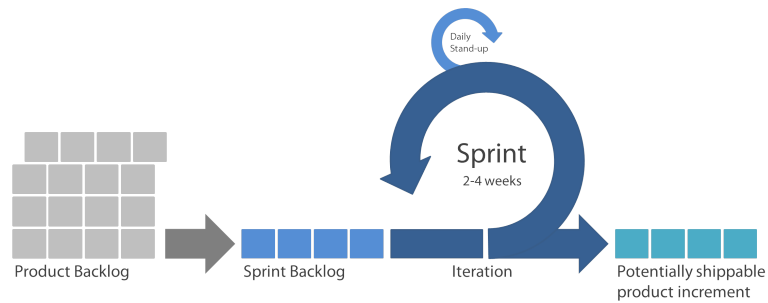


Figura 11: Flusso di lavoro Scrum.

### 2.4.3 Applicazione metodologia

L'azienda ha deciso di applicare il metodo Scrum con il supporto di un *server* VSTS ed ha imposto la durata degli *sprint* a una settimana per seguire il piano di progetto originale. La descrizione dell'effettiva applicazione verrà illustrata nel capitolo successivo.

## 3 Lavoro svolto

### 3.1 Piano di lavoro

Per soddisfare i requisiti presenti nel piano di stage io e il *tutor* abbiamo previsto 320 ore di lavoro, suddivise in settimane da 40 ciascuna. Ho iniziato lo stage Lunedì 20/06/2016 e terminato Venerdì 12/08/2016.

#### 3.1.1 Pianificazione del lavoro

Io e il *tutor* aziendale abbiamo suddiviso il lavoro in 4 settimane, in ciascuna delle quali sarei andato a lavorare 5 giorni, 8 ore al giorno. Oltre al lavoro effettivo, ho riservato alcuni momenti specifici durante la settimana per svolgere le seguenti attività:

- lunedì: pianificazione del lavoro da svolgere durante la settimana;
- al più una giornata: breve colloquio per vedere i risultati raggiunti e/o discutere di eventuali questioni progettuali e/o implementative;
- venerdì: consuntivo e verifica dell'attività settimanale con stesura di un riassunto del lavoro svolto.

In base a quanto precedentemente detto, la pianificazione in termini di quantità di ore di lavoro è stata così distribuita:

Durata in ore		Descrizione dell'attività
40		Analisi dei requisiti per l'applicativo mobile
	8	<ul style="list-style-type: none"><li>• Studio di tecniche di analisi dei requisiti</li><li>• Analisi dei requisiti da implementare nell'applicativo</li><li>• Analisi dei requisiti specifici per piattaforma iOS</li><li>• Analisi dei requisiti specifici per piattaforma Android</li><li>• Analisi dei requisiti specifici per piattaforma Windows 10</li></ul>
	8	
	8	
	8	
	8	
124		Sviluppo completo <i>view</i> dell'applicazione
	76	<ul style="list-style-type: none"><li>• Sviluppo <i>view</i> XAML e C#</li><li>• Sviluppo componenti grafiche native iOS</li><li>• Sviluppo componenti grafiche native Android</li><li>• Sviluppo componenti grafiche native Windows 10</li></ul>
	16	
	16	
	16	
16		Studio delle <i>API</i> ad-hoc
	16	<ul style="list-style-type: none"><li>• Apprendimento di tecniche read/write per <i>API</i></li></ul>
130		Implementazione della logica dell'applicazione
	40	<ul style="list-style-type: none"><li>• Sviluppo <i>model</i></li><li>• Sviluppo <i>viewmodel</i></li><li>• Sviluppo classi <i>helper</i></li></ul>
	50	
	40	
10		Stesura riassunto settimanale e resoconto attività

### 3.1.2 Stesura del piano di lavoro

L'intero piano di lavoro è stato suddiviso in macro attività:

1. Analisi dei requisiti per l'applicativo *mobile*;
2. Sviluppo completo *view* dell'applicazione;
3. Studio delle *API* ad-hoc;
4. Implementazione della logica dell'applicazione.

Sono state inoltre riservate 10 ore distribuite durante tutto lo *stage* per lo svolgimento di altre attività. Queste ore sono necessarie per tenere traccia del lavoro svolto, in modo da avere una comprensione maggiore della sua progressione, e poter tracciare quali attività dovranno essere svolte.


		
Nome	Data d'inizio	Data di fine
• Inizio stage	20/06/16	20/06/16
▣ • Analisi dei requisiti per l'a...	20/06/16	24/06/16
• Studio di tecniche di an...	20/06/16	20/06/16
• Analisi dei requisiti da i...	21/06/16	21/06/16
• Analisi dei requisiti spe...	22/06/16	22/06/16
• Analisi dei requisiti spe...	23/06/16	23/06/16
• Analisi dei requisiti spe...	24/06/16	24/06/16
• Milestone 1	24/06/16	24/06/16
▣ • Sviluppo completo view d...	27/06/16	14/07/16
• Sviluppo view XAML e ...	27/06/16	07/07/16
• Milestone 2	01/07/16	01/07/16
• Sviluppo componenti g...	08/07/16	08/07/16
• Milestone 3	08/07/16	08/07/16
• Sviluppo componenti g...	11/07/16	12/07/16
• Sviluppo componenti g...	13/07/16	14/07/16
▣ • Studio delle API ad-hoc	15/07/16	18/07/16
• Apprendimento di tecn...	15/07/16	18/07/16
• Milestone 4	15/07/16	15/07/16
▣ • Implementazione della lo...	19/07/16	12/08/16
• Sviluppo model	19/07/16	25/07/16
• Milestone 5	22/07/16	22/07/16
• Sviluppo viewmodel	26/07/16	02/08/16
• Milestone 6	29/07/16	29/07/16
• Sviluppo classi aiuto	03/08/16	09/08/16
• Milestone 7	05/08/16	05/08/16
• Testing dell'applicativo	12/08/16	12/08/16
• Milestone 8	12/08/16	12/08/16
• Fine stage	12/08/16	12/08/16

Figura 12: Tabella riassuntiva con la pianificazione delle macro attività.

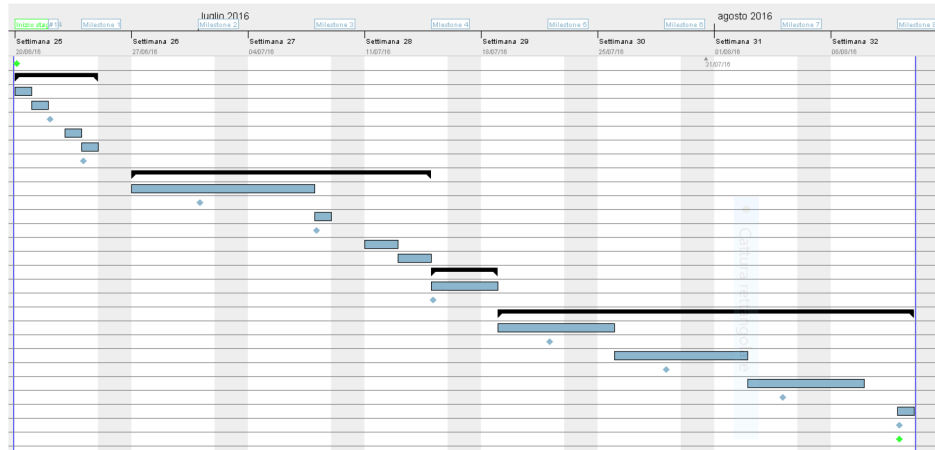


Figura 13: Diagramma di Gantt del piano di stage.

### 3.1.3 Requisiti

Durante la stesura del piano di lavoro io e il *tutor* abbiamo prefissato degli obiettivi per la buona riuscita dello *stage*. I requisiti non sono specifici dell'applicazione ma generali e sono divisi in 3 categorie.

Vengono di seguito riportati i requisiti di progetto a cui si farà riferimento secondo le seguenti notazioni:

- «n» per i requisiti necessari, vincolanti in quanto obiettivi primari;
- «d» per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- «o» per i requisiti opzionali, rappresentanti valore aggiunto non strettamente competitivo.

Inoltre, ad ogni requisito è associato un numero per distinguerli con più facilità. I requisiti sono i seguenti:

- Necessari:
  - «n01»: Comprensione dell'ambiente di lavoro per lo sviluppo (Visual Studio);
  - «n02»: Definizione di quali attività saranno considerate basilari e quali supplementari;
  - «n03»: Acquisizione di familiarità con la programmazione C# e XAML utilizzando Xamarin;
  - «n04»: Acquisizione tecniche di lavoro con *API read/Write*;

- «n05»: Implementazione delle *view* necessarie alle funzioni basilari;
  - «n06»: Implementazione delle funzionalità base;
  - «n07»: Implementazione dell'intera applicazione;
  - «n08»: Stesura resoconto settimanale.
- Desiderabili:
    - «d01»: Acquisizione di familiarità con le dinamiche di *team*;
    - «d02»: Acquisizione di familiarità con le dinamiche di *debug* e *test*;
    - «d03»: Implementazione delle *view* necessarie alle funzionalità supplementari;
    - «d04»: Implementazione delle funzionalità supplementari;
    - «d05»: Integrazione delle funzionalità supplementari nell'applicativo già funzionante;
  - Opzionali:
    - «o01»: Riportare al *team* notizioni utili scoperte;
    - «o02»: Proporre al *team* l'aggiunta di funzionalità non presente nelle applicazioni;
    - «o03»: Implementazione delle funzioni proposte.

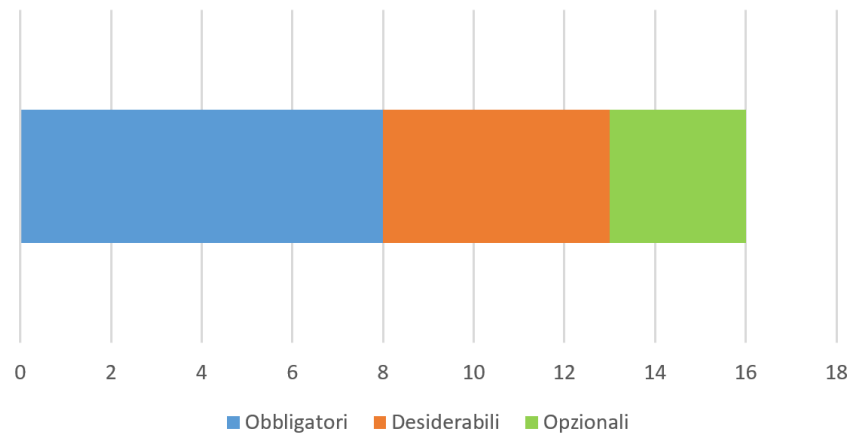


Figura 14: Grafico a barre rappresentante i requisiti suddivisi per tipologia.

Allo scopo di facilitare la verifica, a ogni settimana viene associata una *milestone* di progetto. A ogni *milestone* invece vengono associati più requisiti e la realizzazione di quest'ultimi porta al completamento della *milestone* ad essi associata. Per maggiore chiarezza, nella tabella seguente vengono riportati i legami tra *milestone* e requisiti.



Milestone	Requisiti associati
Milestone 1	n01, n02, n08
Milestone 2, 3, 4	n03, n04, n05, n06, n08
Milestone 5, 6	n05, n06, n08
Milestone 7, 8	n07, n08

## 3.2 Analisi dei requisiti

Secondo il piano di progetto la prima macro attività è l'analisi dei requisiti. Il primo giorno di *stage* abbiamo svolto una riunione con il capo progetto il quale ha spiegato nello specifico quali erano le funzioni che l'applicazione doveva avere. Io e il *tutor* aziendale abbiamo preso appunti, utilizzati poi per svolgere l'analisi dei requisiti.

### 3.2.1 Tecniche utilizzate

Per l'analisi dei requisiti sono state utilizzate le seguenti tecniche:

- Colloquio con il capo progetto;
- Disegno dei casi d'uso;
- Scrittura dei requisiti derivanti dai casi d'uso;
- Scrittura dei requisiti non derivanti dai casi d'uso;
- Classificazione dei requisiti.

Per il disegno dei diagrammi dei casi d'uso ho utilizzato il *software* Astah-Professional, con la licenza per studenti, seguendo lo *standard* UML2.0. I casi d'uso posso avere due attori principali in base al tipo di accesso: cliente o tecnico.

Prima di iniziare la stesura dei requisiti ho imposto una codifica uniforme per la scrittura di quest'ultimi. L'azienda mi ha lasciato libertà su questa scelta, così ho adottato la stessa codifica utilizzata per il progetto di Ingegneria del *Software*. Ogni requisito ha seguito la codifica seguente:

**R[Codice][Importanza][Tipo]**

- **Codice:** codice univoco espresso in modo gerarchico;
- **Importanza:** può assumere i seguenti valori:
  - **N:** necessario;
  - **D:** desiderabile;
  - **O:** opzionale.
- **Tipo:** può assumere i seguenti valori:
  - **F:** funzionale;

- **Q**: di qualità;
- **P**: prestazionale;
- **V**: vincolo.

Questa struttura dà una visione chiara della priorità assegnata ai requisiti.

### 3.2.2 Casi d'uso

I casi d'uso sono rappresentati attraverso alcuni elementi:

- Una numerazione gerarchica;
- Un attore principale;
- Precondizioni;
- Descrizione;
- Postcondizioni.

Di seguito sono presenti i casi d'uso realizzati durante l'analisi dei requisiti.

#### 3.2.2.1 UC1 - Accesso

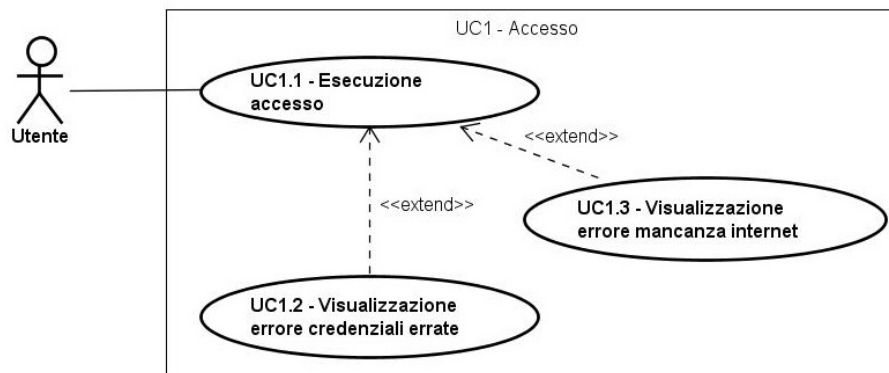


Figura 15: Diagramma *UML* del caso d'uso UC1 - Accesso.

**Attori principali:** Utente generico

**Precondizioni:** L'utente ha aperto l'applicazione

**Descrizione:** L'utente esegue l'accesso se c'è connessione ad internet e se le credenziali sono corrette

**Postcondizioni:** L'utente ha eseguito l'accesso

### 3.2.2.2 UC1.1 - Esecuzione accesso

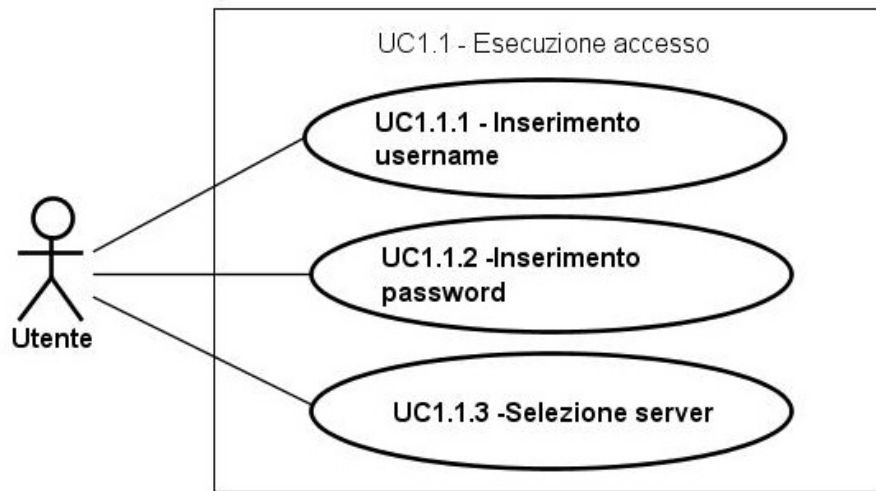


Figura 16: Diagramma *UML* del caso d'uso UC1.1 - Esecuzione accesso.

**Attori principali:** Utente generico

**Precondizioni:** L'utente ha aperto l'applicazione

**Descrizione:** L'utente inserisce i dati e preme il tasto di accesso

**Postcondizioni:** L'utente esegue l'accesso

### 3.2.2.3 UC1.1.1 - Inserimento username

**Attori principali:** Utente generico

**Precondizioni:** L'utente ha aperto l'applicazione

**Descrizione:** L'utente inserisce l'*username*

**Postcondizioni:** L'utente ha inserito l'*username*

### 3.2.2.4 UC1.1.2 - Inserimento password

**Attori principali:** Utente generico

**Precondizioni:** L'utente ha aperto l'applicazione

**Descrizione:** L'utente inserisce la *password*

**Postcondizioni:** L'utente inserisce la *password*

#### **3.2.2.5 UC1.1.3 - Selezione server**

**Attori principali:** Utente generico

**Precondizioni:** L'utente ha aperto l'applicazione

**Descrizione:** L'utente seleziona il *server* a cui connettersi

**Postcondizioni:** L'utente ha selezionato il *server* a cui connettersi

#### **3.2.2.6 UC1.2 - Visualizzazione errore credenziali errate**

**Attori principali:** Utente generico

**Precondizioni:** L'utente ha inserito i dati e ha premuto il tasto di accesso

**Descrizione:** Viene visualizzato un messaggio di errore

**Postcondizioni:** L'utente visualizza il messaggio di errore

#### **3.2.2.7 UC1.3 - Visualizzazione errore mancanza internet**

**Attori principali:** Utente generico

**Precondizioni:** Internet non è presente

**Descrizione:** Viene visualizzato un messaggio di errore

**Postcondizioni:** L'utente visualizza il messaggio di errore

#### **3.2.2.8 UC2 - Visualizzazione schermata principale**

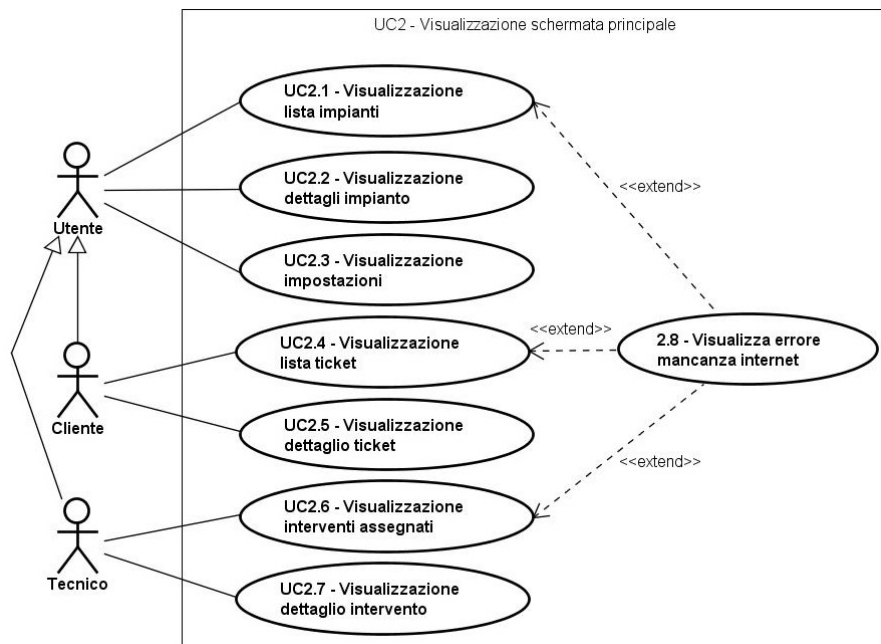


Figura 17: Diagramma *UML* del caso d'uso UC2 - Visualizzazione schermata principale.

**Attori principali:** Utente generico

**Precondizioni:** L'utente ha eseguito l'accesso

**Descrizione:** Viene visualizzato il menu principale ad *hamburger*

**Postcondizioni:** Nessuna

### 3.2.2.9 UC2.1 - Visualizzazione lista impianti

**Attori principali:** Utente generico

**Precondizioni:** L'utente si trova nella schermata principale e gli impianti sono stati scaricati

**Descrizione:** L'utente visualizza la lista di impianti se è presente la connessione ad internet

**Postcondizioni:** Nessuna

### 3.2.2.10 UC2.2 - Visualizzazione dettagli impianto

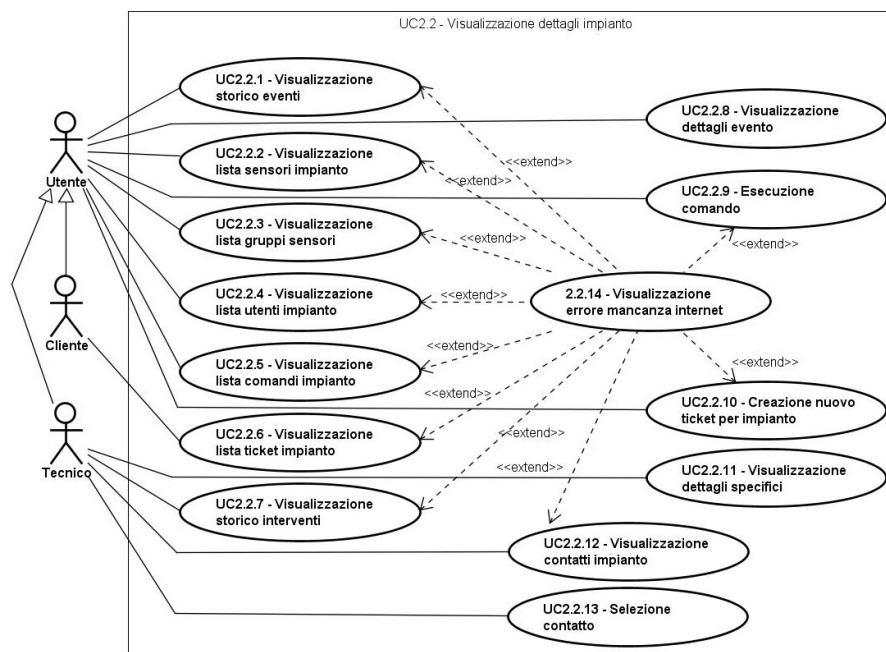


Figura 18: Diagramma *UML* del caso d'uso UC2.2 - Visualizzazione dettagli impianto.

**Attori principali:** Utente generico

**Precondizioni:** L'utente ha selezionato un impianto

**Descrizione:** L'utente visualizza nel dettaglio l'impianto

**Postcondizioni:** Vengono scaricati i dati dettagliati dell'impianto

### 3.2.2.11 UC2.2.1 - Visualizzazione storico eventi

**Attori principali:** Utente generico

**Precondizioni:** L'utente è nella schermata di dettaglio dell'impianto e lo storico eventi è scaricato

**Descrizione:** L'utente visualizza lo storico degli eventi dell'impianto (data, ora, tipo, ecc..)

**Postcondizioni:** Nessuna

### 3.2.2.12 UC2.2.2 - Visualizzazione lista sensori

**Attori principali:** Utente generico

**Precondizioni:** L'utente è nella schermata di dettaglio dell'impianto e i sensori

sono scaricati

**Descrizione:** L'utente visualizza i sensori dell'impianto (tipo, immagine, stato, nome)

**Postcondizioni:** Nessuna

#### 3.2.2.13 UC2.2.3 - Visualizzazione lista gruppi

**Attori principali:** Utente generico

**Precondizioni:** L'utente è nella schermata di dettaglio dell'impianto e i gruppi di sensori sono scaricati

**Descrizione:** L'utente visualizza i gruppi di sensori dell'impianto (tipo, immagine, stato, nome)

**Postcondizioni:** Nessuna

#### 3.2.2.14 UC2.2.4 - Visualizzazione lista utenti impianto

**Attori principali:** Utente generico

**Precondizioni:** L'utente è nella schermata di dettaglio dell'impianto e gli utenti sono scaricati

**Descrizione:** L'utente visualizza gli utenti dell'impianto (codice, nome)

**Postcondizioni:** Nessuna

#### 3.2.2.15 UC2.2.5 - Visualizzazione lista comandi impianto

**Attori principali:** Utente generico

**Precondizioni:** L'utente è nella schermata di dettaglio dell'impianto e la lista di comandi è scaricata

**Descrizione:** L'utente visualizza la lista di comandi dell'impianto (tipo, immagine, stato, nome)

**Postcondizioni:** Nessuna

#### 3.2.2.16 UC2.2.6 - Visualizzazione lista *ticket* impianto

**Attori principali:** Cliente

**Precondizioni:** Il cliente è nella schermata di dettaglio dell'impianto e i *ticket* sono scaricati

**Descrizione:** L'utente visualizza i *ticket* dell'impianto (stato, titolo)

**Postcondizioni:** Nessuna

### 3.2.2.17 UC2.2.7 - Visualizzazione storico interventi

**Attori principali:** Tecnico

**Precondizioni:** Il tecnico è nella schermata di dettaglio dell'impianto e gli interventi sono scaricati

**Descrizione:** Il tecnico visualizza gli interventi dell'impianto (stato, titolo, codice impianto, indirizzo)

**Postcondizioni:** Nessuna

### 3.2.2.18 UC2.2.8 - Visualizzazione dettaglio evento

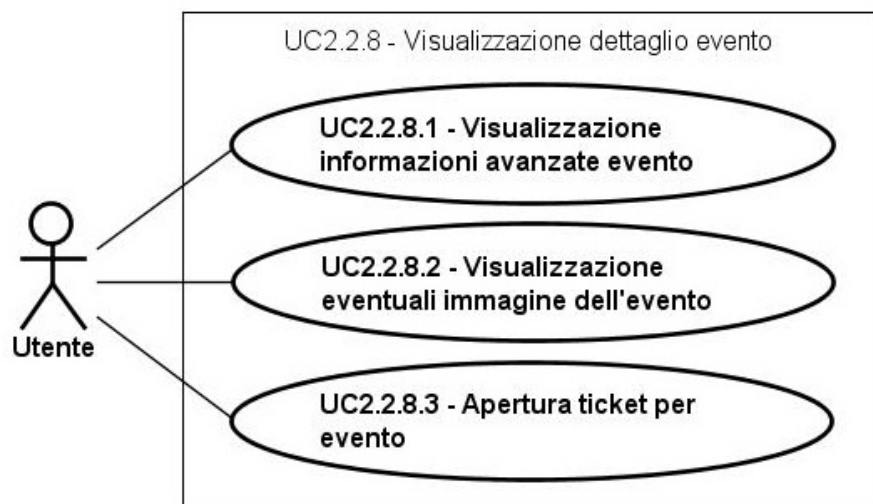


Figura 19: Diagramma *UML* del caso d'uso UC2.8 - Visualizzazione dettaglio storico.

**Attori principali:** Utente generico

**Precondizioni:** L'utente ha selezionato un evento dallo storico

**Descrizione:** L'utente visualizza i dettagli dello storico

**Postcondizioni:** Nessuna

### 3.2.2.19 UC2.2.8.1 - Visualizzazione informazioni avanzate evento

**Attori principali:** Utente generico

**Precondizioni:** L'utente è nella pagina di dettaglio dell'evento



**Descrizione:** L'utente visualizza informazioni avanzate dell'evento (codice originale, immagine, ecc..)

**Postcondizioni:** Nessuna

#### **3.2.2.20 UC2.2.8.2 - Visualizzazione eventuali immagini dell'evento**

**Attori principali:** Utente generico

**Precondizioni:** L'utente è nella pagina di dettaglio dell'evento

**Descrizione:** Se presenti, l'utente visualizza l'immagine scattata dalla telecamera durante quell'evento

**Postcondizioni:** Nessuna

#### **3.2.2.21 UC2.2.8.3 - Apertura *ticket* per evento**

**Attori principali:** Utente generico

**Precondizioni:** L'utente è nella pagina di dettaglio dell'evento

**Descrizione:** L'utente crea un nuovo *ticket* riferito a quell'evento

**Postcondizioni:** La segnalazione viene inviata

#### **3.2.2.22 UC2.2.9 - Esecuzione comando**

**Attori principali:** Utente generico

**Precondizioni:** L'utente seleziona un comando dalla lista

**Descrizione:** L'utente inserisce il codice di sicurezza

**Postcondizioni:** Il comando viene inviato

#### **3.2.2.23 UC2.2.10 - Creazione nuovo *ticket* per impianto**

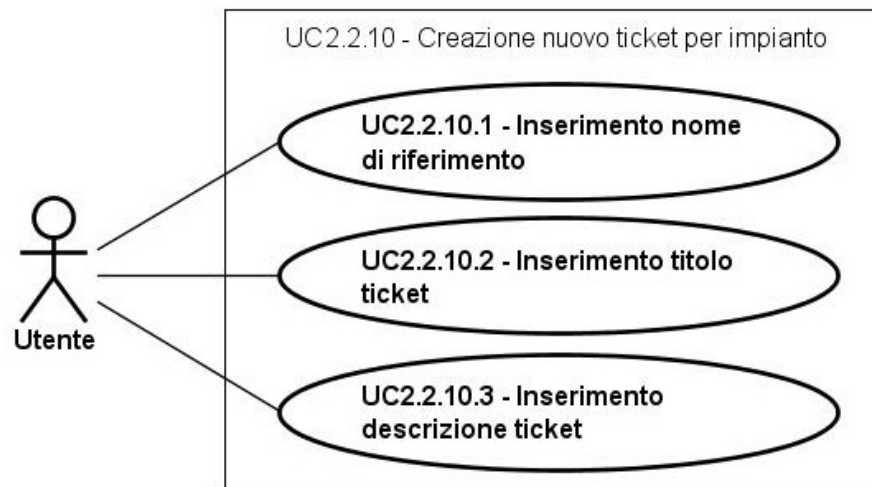


Figura 20: Diagramma *UML* del caso d'uso UC2.2.10 - Creazione nuovo *ticket* per impianto.

**Attori principali:** Utente generico

**Precondizioni:** L'utente è nella schermata di dettaglio dell'impianto

**Descrizione:** L'utente inserisce i dati per il *ticket*

**Postcondizioni:** Il *ticket* viene inviato

#### 3.2.2.24 UC2.2.10.1 - Inserimento nome di riferimento

**Attori principali:** Utente generico

**Precondizioni:** L'utente è nella schermata di creazione *ticket*

**Descrizione:** L'utente inserisce il nome di riferimento

**Postcondizioni:** L'utente ha inserito il nome di riferimento

#### 3.2.2.25 UC2.2.10.2 - Inserimento titolo *ticket*

**Attori principali:** Utente generico

**Precondizioni:** L'utente è nella schermata di creazione *ticket*

**Descrizione:** L'utente inserisce il titolo

**Postcondizioni:** L'utente ha inserito il titolo

#### 3.2.2.26 UC2.2.10.3 - Inserimento descrizione

**Attori principali:** Utente generico

**Precondizioni:** L'utente è nella schermata di creazione *ticket*

**Descrizione:** L'utente inserisce la descrizione

**Postcondizioni:** L'utente ha inserito la descrizione

### 3.2.2.27 UC2.2.11 - Visualizzazione dettagli specifici

**Attori principali:** Tecnico

**Precondizioni:** Il tecnico è nella schermata di dettaglio dell'impianto

**Descrizione:** Il tecnico visualizza i dettagli specifici dell'impianto

**Postcondizioni:** Nessuna

### 3.2.2.28 UC2.2.12 - Visualizzazione contatti impianto

**Attori principali:** Utente generico

**Precondizioni:** Il tecnico è nella schermata di dettaglio dell'impianto e i contatti sono scaricati

**Descrizione:** Il tecnico visualizza i contatti dell'impianto

**Postcondizioni:** Nessuna

### 3.2.2.29 UC2.2.13 - Selezione contatto

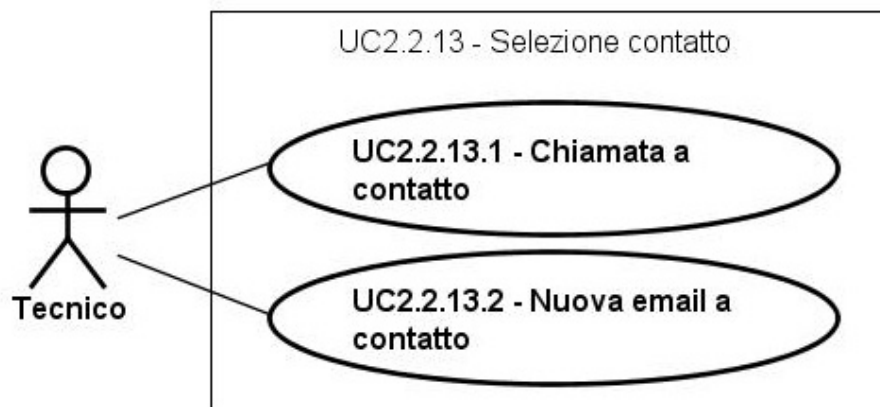


Figura 21: Diagramma *UML* del caso d'uso UC2.2.13 - Selezione contatto.

**Attori principali:** Tecnico

**Precondizioni:** Il tecnico seleziona un contatto nella lista di contatti

**Descrizione:** Il tecnico contatta l'utente selezionato

**Postcondizioni:** Il tecnico torna all'applicazione

#### **3.2.2.30 UC2.2.13.1 - Chiamata a contatto**

**Attori principali:** Tecnico

**Precondizioni:** Il contatto selezionato ha un numero di cellulare

**Descrizione:** Viene aperta l'applicazione per le chiamate

**Postcondizioni:** Il tecnico torna all'applicazione

#### **3.2.2.31 UC2.2.13.2 - Nuova *email* a contatto**

**Attori principali:** Tecnico

**Precondizioni:** Il contatto selezionato ha un indirizzo *email*

**Descrizione:** Viene aperta l'applicazione per la posta elettronica

**Postcondizioni:** Il tecnico torna all'applicazione

#### **3.2.2.32 UC2.2.14 - Visualizzazione errore mancanza internet**

**Attori principali:** Utente generico

**Precondizioni:** Internet non è presente

**Descrizione:** Viene visualizzato un messaggio di errore

**Postcondizioni:** L'utente visualizza il messaggio di errore

#### **3.2.2.33 UC2.3 - Visualizzazione impostazioni**

**Attori principali:** Utente generico

**Precondizioni:** L'utente si trova nella schermata principale

**Descrizione:** L'utente visualizza le impostazioni dei filtri e del *GPS* e può modificarle

**Postcondizioni:** L'utente può aver modificato le impostazioni

#### **3.2.2.34 UC2.4 - Visualizzazione lista *ticket***

**Attori principali:** Cliente

**Precondizioni:** L'utente si trova nella schermata principale, la lista di *ticket* è stata scaricata

**Descrizione:** L'utente visualizza la lista di *ticket*

**Postcondizioni:** Nessuna

### 3.2.2.35 UC2.5 - Visualizzazione dettaglio *ticket*

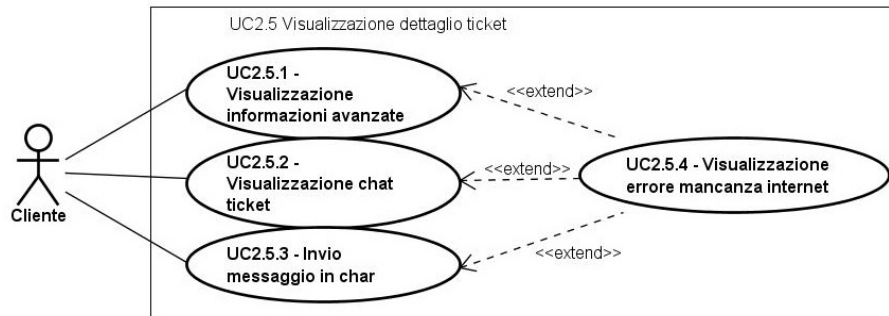


Figura 22: Diagramma *UML* del caso d'uso UC2.5 - Visualizzazione dettaglio *ticket*.

**Attori principali:** Cliente

**Precondizioni:** Il cliente ha selezionato un *ticket*

**Descrizione:** Il cliente visualizza i dettagli del *ticket* e la *chat*

**Postcondizioni:** Il cliente può aver inviato un messaggio in *chat*

### 3.2.2.36 UC2.5.1 - Visualizzazione informazioni avanzate

**Attori principali:** Cliente

**Precondizioni:** Il cliente è nella schermata di dettaglio del *ticket*

**Descrizione:** Il cliente visualizza le informazioni avanzate del *ticket*

**Postcondizioni:** Nessuna

### 3.2.2.37 UC2.5.2 - Visualizzazione *chat ticket*

**Attori principali:** Cliente

**Precondizioni:** Il cliente è nella schermata di dettaglio del *ticket*

**Descrizione:** Il cliente visualizza la *chat* del *ticket*

**Postcondizioni:** Nessuna

### 3.2.2.38 UC2.5.3 - Invio messaggio in *chat*

**Attori principali:** Cliente

**Precondizioni:** Il cliente è nella schermata di dettaglio del *ticket*

**Descrizione:** Il cliente inserisce il testo del messaggio ed invia

**Postcondizioni:** Il cliente ha inviato un messaggio nella *chat*

### 3.2.2.39 UC2.6 - Visualizzazione interventi assegnati

**Attori principali:** Tecnico

**Precondizioni:** Il tecnico si trova nella schermata principale, la lista di interventi assegnati è stata scaricata

**Descrizione:** Il tecnico visualizza la lista di interventi a lui assegnati

**Postcondizioni:** Nessuna

### 3.2.2.40 UC2.7 - Visualizzazione dettaglio intervento

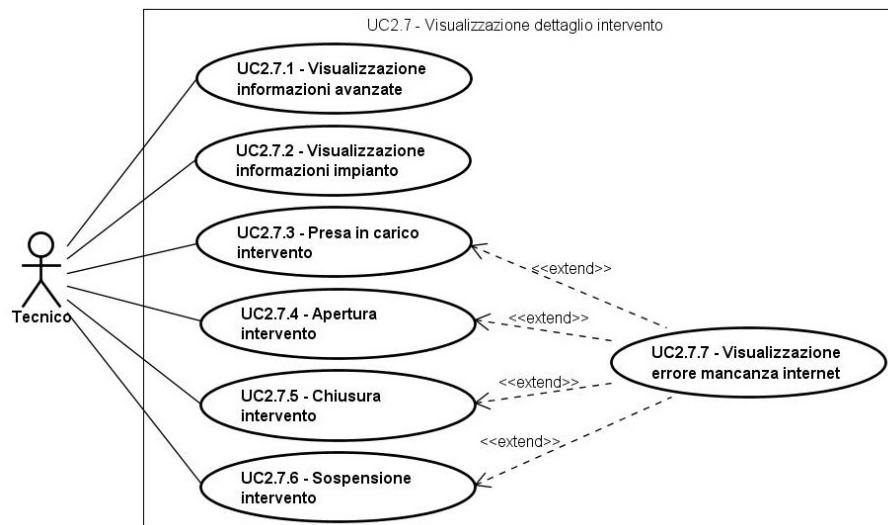


Figura 23: Diagramma *UML* del caso d'uso UC2.7 - Visualizzazione dettaglio intervento.

**Attori principali:** Tecnico

**Precondizioni:** Il tecnico ha selezionato un intervento

**Descrizione:** Il tecnico visualizza i dettagli dell'intervento

**Postcondizioni:** Il tecnico può aver effettuato operazioni sull'intervento

### 3.2.2.41 UC2.7.1 - Visualizzazione informazioni avanzate

**Attori principali:** Tecnico

**Precondizioni:** Il tecnico è nella schermata di dettaglio dell'intervento

**Descrizione:** Il tecnico visualizza le informazioni avanzate dell'intervento

**Postcondizioni:** Nessuna

#### **3.2.2.42 UC2.7.2 - Visualizzazione informazioni impianto**

**Attori principali:** Tecnico

**Precondizioni:** Il tecnico è nella schermata di dettaglio dell'intervento

**Descrizione:** Il tecnico visualizza le informazioni dell'impianto associato dell'intervento

**Postcondizioni:** Nessuna

#### **3.2.2.43 UC2.7.3 - Presa in carico intervento**

**Attori principali:** Tecnico

**Precondizioni:** Il tecnico è nella schermata di dettaglio dell'intervento

**Descrizione:** Il tecnico prende in carico l'intervento

**Postcondizioni:** Il tecnico ha preso in carico l'intervento

#### **3.2.2.44 UC2.7.4 - Apertura intervento**

**Attori principali:** Tecnico

**Precondizioni:** Il tecnico è nella schermata di dettaglio dell'intervento

**Descrizione:** Il tecnico apre l'intervento

**Postcondizioni:** Il tecnico ha aperto l'intervento

#### **3.2.2.45 UC2.7.5 - Chiusura intervento**

**Attori principali:** Tecnico

**Precondizioni:** Il tecnico è nella schermata di dettaglio dell'intervento

**Descrizione:** Il tecnico chiude l'intervento

**Postcondizioni:** Il tecnico ha chiuso l'intervento

#### **3.2.2.46 UC2.7.6 - Sospensione intervento**

**Attori principali:** Tecnico

**Precondizioni:** Il tecnico è nella schermata di dettaglio dell'intervento

**Descrizione:** Il tecnico sospende l'intervento

**Postcondizioni:** Il tecnico ha sospeso l'intervento

#### 3.2.2.47 UC2.7.7 - Visualizzazione errore mancanza internet

**Attori principali:** Tecnico

**Precondizioni:** Internet non è presente

**Descrizione:** Viene visualizzato un messaggio di errore

**Postcondizioni:** Il tecnico visualizza il messaggio di errore

#### 3.2.2.48 UC2.8 - Visualizzazione errore mancanza internet

**Attori principali:** Utente generico

**Precondizioni:** Internet non è presente

**Descrizione:** Viene visualizzato un messaggio di errore

**Postcondizioni:** L'utente visualizza il messaggio di errore

### 3.2.3 Suddivisione dei requisiti

Una volta scritti tutti i requisiti, li ho classificati seguendo la codifica descritta in precedenza.

#### 3.2.3.1 Requisiti funzionali

I requisiti funzionali riguardano le specifiche del prodotto. Il soddisfacimento di uno di essi equivale all'implementazione di una funzionalità.

Codice	Descrizione
R[1][N][F]	Accesso
R[1.1][N][F]	<i>Download</i> lista server
R[1.2][N][F]	Richiesta di accesso al server selezionato
R[1.3][N][F]	Salvataggio credenziali
R[1.4][N][F]	<i>Logout</i>
R[1.5][N][F]	Accesso automatico
R[2][N][F]	Menu principale
R[2.1][N][F]	Lista impianti utente
R[2.2][D][F]	Lista <i>ticket</i> utente
R[2.3][D][F]	Lista interventi tecnico
R[2.4][N][F]	Impostazioni
R[2.4.1][N][F]	Opzioni di filtraggio
R[2.4.2][D][F]	Impostazioni <i>GPS</i>
R[3][N][F]	Dettaglio impianto selezionato
R[3.1][N][F]	Lista storico eventi
R[3.2][N][F]	Lista gruppi sensori



R[3.3][N][F]	Lista sensori
R[3.4][N][F]	Lista utenti
R[3.5][N][F]	Lista comandi
R[3.6][N][F]	Creazione <i>ticket</i> per impianto
R[3.7][N][F]	Visualizzazione specifiche impianto
R[3.7.1][D][F]	Visualizzazione informazione tecniche
R[3.7.2][D][F]	Visualizzazione contatti impianto
R[3.7.2.1][D][F]	Chiamata a contatto selezionato
R[3.7.3.2][D][F]	<i>Email</i> a contatto selezionato
R[3.7.3][D][F]	Visualizzazione <i>ticket</i> impianto
R[3.7.4][D][F]	Visualizzazione interventi impianto
R[4][N][F]	Dettaglio <i>ticket</i> selezionato
R[4.1][N][F]	Visualizzazione informazioni aggiuntive
R[4.2][D][F]	Visualizzazione <i>chat</i> del <i>ticket</i>
R[4.3][D][F]	Invio nuovo messaggio in <i>chat</i>
R[5][D][F]	Dettaglio intervento selezionato
R[5.1][D][F]	Visualizzazione informazioni aggiuntive
R[5.2][D][F]	Invio presa in carico intervento
R[5.3][D][F]	Invio apertura intervento
R[5.4][D][F]	Invio chiusura intervento
R[5.5][D][F]	Invio sospensione intervento
R[6][N][F]	Dettaglio storico selezionato
R[6.1][N][F]	Visualizzazione informazioni aggiuntive
R[6.2][N][F]	Creazione <i>ticket</i> per l'evento
R[7][N][F]	Esecuzione comando selezionato
R[7.1][N][F]	Chiedere codice di sicurezza
R[7.2][N][F]	Invio comando al <i>server</i>
R[8][D][F]	Gestione posizione <i>GPS</i> tecnici
R[8.1][D][F]	Rilevazione periodica
R[8.2][D][F]	Rilevazione a richiesta del <i>server</i>
R[8.3][D][F]	Invio posizione al <i>server</i>
R[9][N][F]	Gestione notifiche
R[9.1][N][F]	Gestione notifiche Windows 10
R[9.2][N][F]	Gestione notifiche Android
R[9.3][N][F]	Gestione notifiche iOS

Tabella 1: Tabella dei requisiti funzionali scritti con il *tutor* durante l'analisi dei requisiti

### 3.2.3.2 Requisiti di vincolo

I requisiti di vincolo rappresentano i vincoli che devono essere soddisfatti dal prodotto e che esulano dalle sue caratteristiche funzionali.

Codice	Descrizione
R[10][N][V]	Utilizzo di Xamarin.Forms
R[11][N][V]	Rispetto linee guida di ogni piattaforma
R[12][N][V]	Funzionamento corretto in base al sistema operativo
R[12.1][N][V]	Funzionamento corretto su Windows 10
R[12.2][N][V]	Funzionamento corretto su iOS 8.1 o successivi
R[12.3][N][V]	Funzionamento corretto su Android 4.4 o successivi

Tabella 2: Tabella dei requisiti di vincolo scritti con il *tutor* durante l'analisi dei requisiti

### 3.2.3.3 Requisiti di qualità

I requisiti di qualità specificano le operazioni da compiere per far raggiungere al prodotto il livello qualitativo richiesto.

Codice	Descrizione
R[13][N][Q]	Qualità del codice
R[13.1][N][Q]	Il codice deve seguire le linee guida
R[13.2][N][Q]	Le parti di difficili devono essere commentate
R[14][N][Q]	Qualità del progetto
R[14.1][N][Q]	Il progetto deve seguire la metodologia Scrum
R[14.2][N][Q]	Per l'organizzazione vanno utilizzati i servizi Visual Studio Team Services
R[14.3][N][Q]	Il codice deve essere sottoposto a controllo codice sorgente

Tabella 3: Tabella dei requisiti di qualità scritti con il *tutor* durante l'analisi dei requisiti

### 3.2.3.4 Requisiti prestazionali

I requisiti prestazionali garantiscono, se soddisfatti, un livello prestazionale maggiore al prodotto finito. Non rappresentano funzionalità aggiuntive ma migliorano stabilità e velocità di quelle già implementate.

Codice	Descrizione
R[15][N][Q]	Avvio dell'applicazione in meno di 30 secondi anche in caso di connessione molto lenta

Tabella 4: Tabella dei requisiti prestazionali scritti con il *tutor* durante l'analisi dei requisiti

### 3.2.4 Considerazioni sui requisiti

Come si può vedere dalle tabelle precedenti non ho previsto requisiti opzionali data la durata limitata dello *stage*.

Ho inserito un unico requisito prestazionale, riguardante il tempo di avvio, a causa di alcune restrizioni legate alla piattaforma iOS, la quale forza la chiusura delle applicazioni, qualora esse non si avviino entro un tempo prestabilito di 30 secondi.

## 3.3 Vincoli di progetto

I vincoli tecnologici e di dominio hanno imposto l'utilizzo di alcune tecnologie e strumenti.

### 3.3.1 Dal progetto

Alcuni vincoli tecnologici sono dettati dalla natura del progetto. Essi erano noti e definiti a priori, già in sede del colloquio di *stage*. L'azienda aveva infatti richiesto lo sviluppo dell'applicazione attraverso *Xamarin.Forms* in modo da garantirne lo sviluppo su tutte e 3 le piattaforme presenti. Lo sviluppo per *Windows 10* ha imposto l'utilizzo dello stesso sistema operativo anche per la compilazione, mentre, da la necessità di *OSX* per la compilazione remota sulla piattaforma *iOS* è stato utilizzato un *Mac Mini* collegato in rete.

Abbiamo scelto l'*IDE Visual Studio 2015* per lo sviluppo completo, in parte per la presenza dell'applicativo per *Windows 10* e in parte perché l'azienda, avendo un abbonamento *Microsoft Developer Network (MSDN)*, ha *software* e supporto completi.

### 3.3.2 Dai processi aziendali

Nonostante io sia stato affiancato da una sola persona durante l'intero progetto, l'azienda ha imposto di seguire la metodologia (Scrum) e gli strumenti (VSTS) utilizzati negli altri progetti anche per questo. Grazie all'abbonamento MSDN, l'azienda ha a disposizione tutti i servizi Visual Studio Team Service i quali sono stati installati in un *server* dedicato. Sono stati utilizzati diversi servizi disponibili nella *suite*:

- Il controllo del codice sorgente, utilizzando Team Foundation Version Control rispetto a Git per la centralizzazione della gestione piuttosto che distribuita;
- La *continuous integration*, impostata per la compilazione e l'esecuzione di test ad ogni modifica del codice;

- La piattaforma *web* per la gestione del progetto con metodologia Scrum sulla quale abbiamo creato le attività da svolgere.

Per le comunicazione invece, abbiamo utilizzato Telegram per la sua semplicità e portabilità. Alla fine di ogni *sprint* ho creato un resoconto delle attività svolte e quelle ancora da svolgere, utile durante le riunioni con il capo progetto.



Figura 24: Logo di Telegram.

## 3.4 Studio delle tecnologie

### 3.4.1 Linguaggi

Gli unici linguaggi utilizzati sono stati: *C#* e *XAML*. Con entrambi avevo familiarità in quanto studiati durante gli studi precedenti al percorso universitario, ma l'utilizzo in ambito *mobile* multiplatforma era per me nuovo, in quanto abituato allo sviluppo *desktop* con la tecnologia Windows Presentation Foundation (WPF).

Conoscevo anche il *pattern* MVVM, avendolo utilizzato durante lo sviluppo di alcune applicazioni WPF. Studiando però le *best practice* suggerite dalla documentazione Xamarin ho trovato leggere differenze rispetto a quel che conoscevo. Le più importanti sono:

- I *command* sono inizializzati nel costruttore e non a ogni richiesta;
- I *viewmodel* non vengono creati tutti all'apertura del programma ma vengono inizializzati solo quando server.

Queste differenze sono dovute alla differenza di calcolo e memoria tra PC e *smartphone*.

### 3.4.2 Progetti nativi

Nonostante il codice logico sia condiviso tra le piattaforme ci sono alcuni elementi che ho dovuto studiare separatamente per ogni piattaforma:

- L'abilitazione dei permessi per le funzionalità avanzate (GPS, scrittura su disco, chiamate, ecc);
- La gestione delle immagini;

- Le modifiche ad elementi grafici;
- I manifesti (nome, versione, ect);
- Le notifiche;
- La gestione degli *store*.

### 3.4.3 Ambienti di sviluppo

Ho utilizzato come unico ambiente di sviluppo *Visual Studio 2015*. Avendo lavorato con WPF per molto tempo, questo *IDE* è per me molto familiare, per questo non ho dovuto impiegare del tempo per apprendere il suo utilizzo.

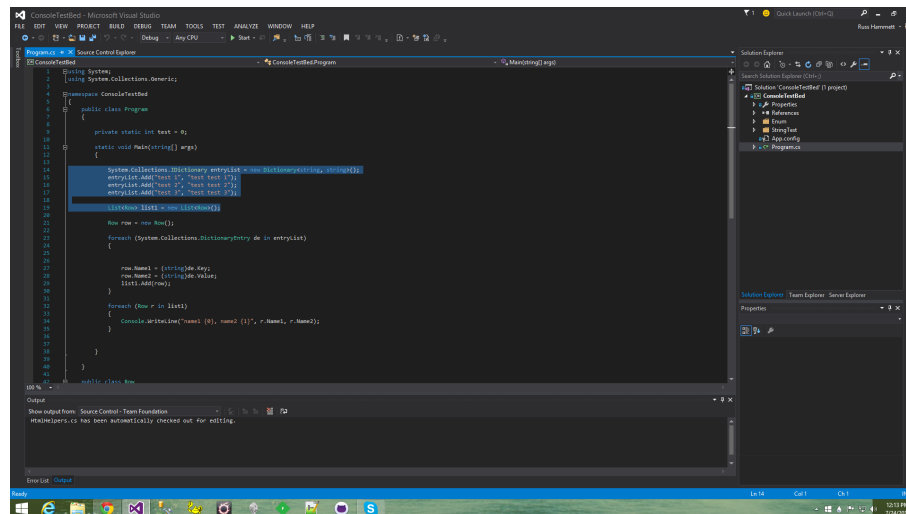


Figura 25: Finestra di esempio di Visual Studio.

Per modifiche veloci invece ho utilizzato come *editor* di testo Visual Studio Code, anch'esso da me ben conosciuto.

### 3.4.4 Strumenti di *team*

Per quanto riguarda gli strumenti di collaborazione in *team* ho avuto esperienze diverse con i vari servizi:

- **Controllo codice sorgente:** L'azienda ha imposto l'utilizzo di TFVC come sistema di controllo del codice sorgente. I concetti base del funzionamento sono simili a quelli di Git, già utilizzato durante il progetto di Ingegneria del *Software*, quindi non ho avuto particolari difficoltà nell'apprendere questa nuova tecnologia;

- **Continuous integration (CI):** il *server* per l'integrazione continua è stato impostato dal mio *tutor* aziendale. Nonostante questo non avrei avuto problemi in quanto ho avuto già esperienze con la *CI* durante il progetto di Ingegneria del *Software*;
- **Portale *web* per la gestione Scrum:** Questo portale per me era un strumento completamente sconosciuto, ho passato alcune ore a studiare i principi di Scrum e successivamente come applicarli sul portale *web*. Fortunamente l'applicativo *online* è molto intuitivo quindi non ho fatto molta fatica. Il *tutor* inoltre mi ha aiutato a prendere familiarità con lo strumento più velocemente.

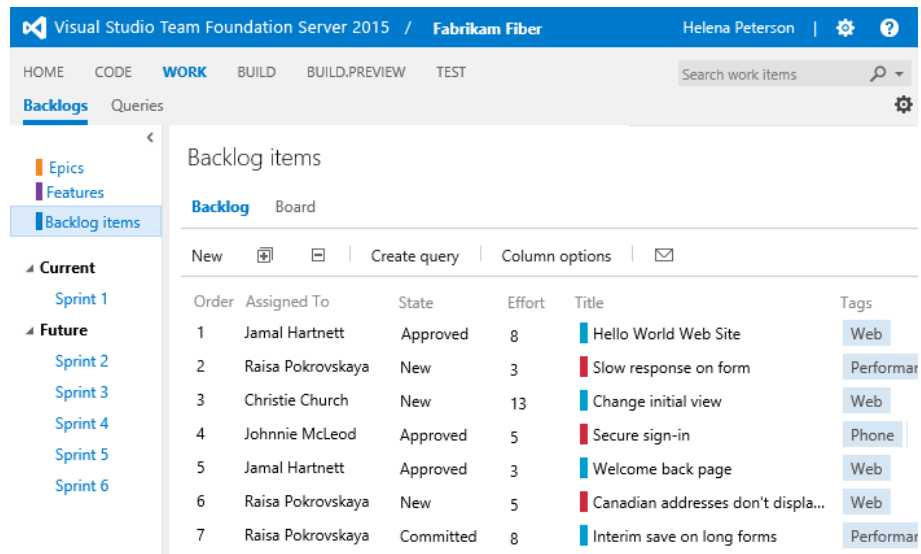


Figura 26: Finestra di esempio del portale *web* Visual Studio Team Services per la gestione di un progetto Scrum.

### 3.4.5 *Framework* e librerie

L'unico *framework* usato è Xamarin.Forms. Non conoscevo bene questa tecnologia, avevo solo seguito qualche guida in passato su Xamarin.Android. Per questo ho studiato da zero il funzionamento attraverso la documentazione ufficiale, un libro e svariati esempi. In particolare ho analizzato in modo approfondito l'esempio ufficiale dell'applicazione CRM. Da questo esempio ho copiato la struttura base per iniziare lo sviluppo nella nuova applicazione PointSecurity Mobile in quanto esperienza e tempo non mi avrebbero permesso di creare un'applicazione di qualità totalmente da solo.



Figura 27: Copertina del libro ufficiale per lo studio di *Xamarin.Forms*.

## 3.5 Sviluppo

### 3.5.1 Struttura dei progetti

Creando un nuovo progetto di tipo Xamarin.Forms in Visual Studio vengono generati 4 progetti, uno per ogni piattaforma e una PCL per il codice da condividere *cross-platform*.

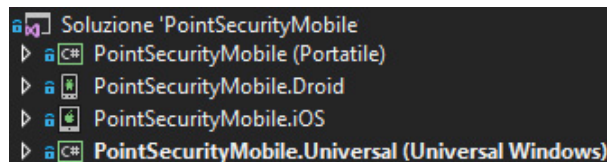


Figura 28: Progetti generati da Visual Studio.

La differenza maggiore fra i progetti, oltre agli elementi nativi come i file *plist* di Apple o la *main activity* di Android, è la presenza di cartelle e *file* per le funzioni avanzate del sistema operativo. Infatti i servizi come le notifiche

*push*, la geolocalizzazione e l'accesso alle chiamate vanno gestite separatamente in quanto Xamarin non è riuscita a creare un *wrapper* per funzioni gestite in modo così diverso tra le piattaforme. Per questo motivo per l'utilizzo di queste funzioni va creata un'interfaccia con i metodi desiderati, la cui implementazione viene fatta in modo diverso, in nativo, nei diversi progetti.

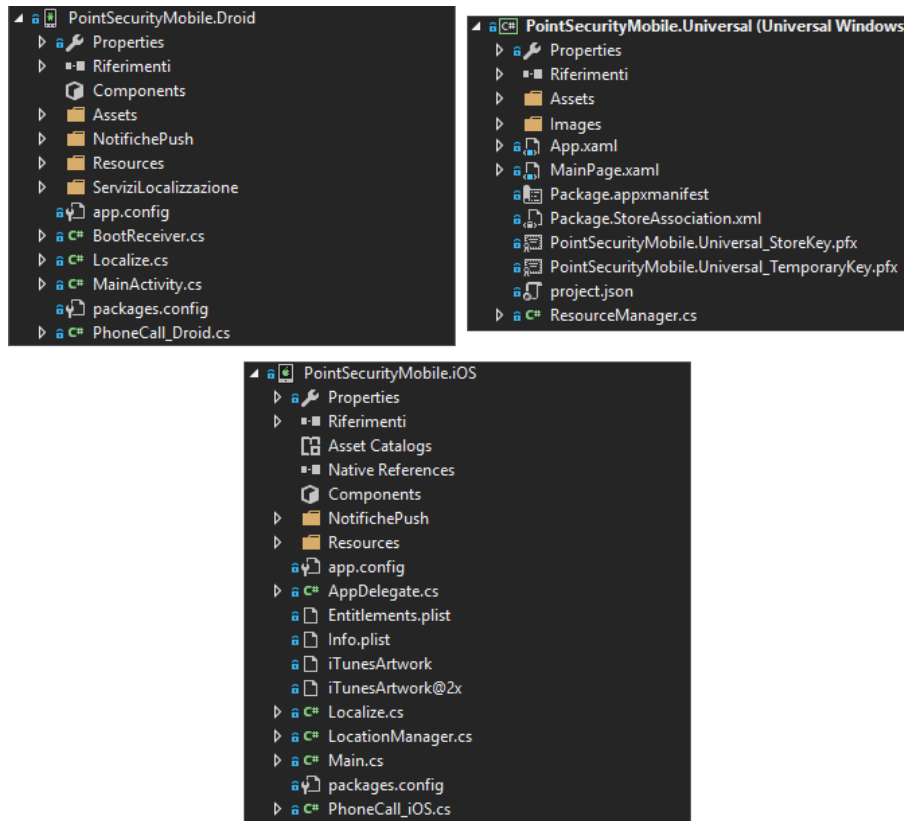


Figura 29: Cartella dei tre progetti Xamarin.Forms finiti.

### 3.5.2 Funzioni base

Le funzioni base sono descritte nella tabella dei requisiti come necessari. Nella sezione seguente descriverò il lavoro da me portato a termine per soddisfare questi requisiti.

#### 3.5.2.1 Richieste alle *API web*

La nuova applicazione usa lo stesso *backend* dell'applicazione che vuole sostituire. Questo ha permesso di riutilizzare la PCL per la comunicazione con le *API web* creata per l'applicazione attuale, anch'essa sviluppata con Xama-



rin.Forms. La libreria è stata però opportunamente modificata, aggiornando le librerie crittografiche e aggiungendo i metodi per sfruttare le *API* create per le nuove funzioni. I dati inviati sono tutti in JSON (*JavaScript Object Notation*), un formato molto popolare per rappresentare dati strutturati, facilmente comprensibile sia dall'uomo che dalle macchine.

L'avanzamento nello sviluppo tra l'applicazione e le *API web* è stato parallelo. La creazione di una nuova *API* da parte del *tutor* comportava lo sviluppo da parte mia di un nuovo metodo nella PCL per interfacciarsi con essa.

### 3.5.2.2 Scrittura su disco

La scrittura su disco serve solamente per memorizzare le credenziali di accesso se l'utente lo desidera. Il processo è stato semplificato da Xamarin nelle ultime versioni del suo *framework* fornendo una PCL per l'interazione con la memoria dei dispositivi. L'unica operazione da effettuare è l'apertura di *stream* di lettura o di scrittura, la libreria esterna si preoccupa di richiamare le *API* native.

All'apertura dell'applicazione essa cerca se sono presenti delle credenziali salvate su disco, le interpreta e tenta l'accesso al *server web*. Se tutto va a buon fine l'utente visualizza direttamente il menu principale. Se invece uno dei seguenti casi si verifica l'utente visualizza la schermata di accesso:

- *File* delle credenziali non trovato;
- *File* corrotto;
- Assenza di connessione ad *internet*;
- *Server offline*;
- Credenziali non più valide (utente eliminato o *password* cambiata).

In tutti i casi, eccetto il primo, l'utente visualizza un messaggio di errore che descrive il problema. In caso di *logout* da parte dell'utente, il *file* viene semplicemente cancellato e l'utente viene rimandato alla pagina di accesso.

### 3.5.2.3 Design

Come accennato in precedenza, per la realizzazione della struttura base ho preso spunto dall'esempio CRM di Xamarin. Esso segue le linee guida di ogni sistema operativo: per la schermata principale utilizza un *menu a tab* in iOS e uno laterale (*hamburger menu*) per Android e Windows 10. Per raggiungere questo obiettivo ho creato due tipi diversi di *view*, iniziando quella corretta in base al sistema operativo rilevato.

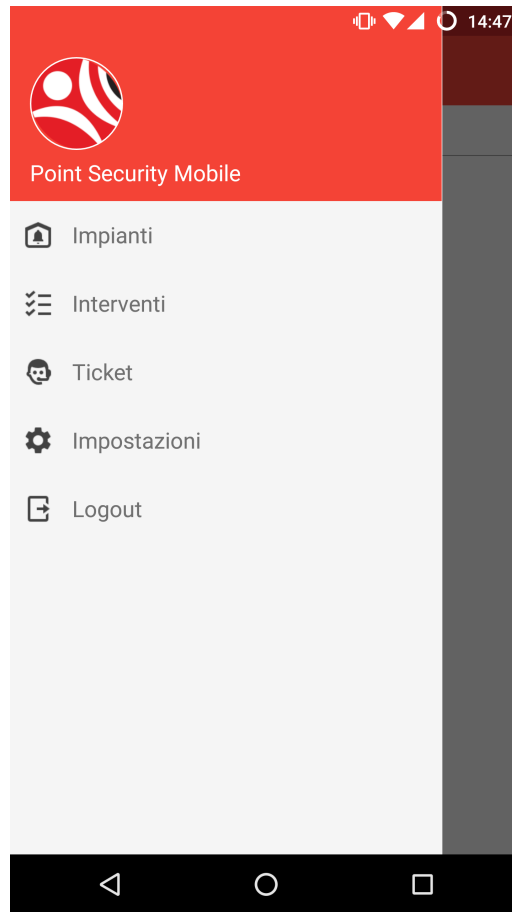


Figura 30: *Screenshot* del menu principale (Android).

Il colore principale dell'azienda è il rosso, ho deciso quindi di rendere i vari menu e barre di navigazione rossi e lo sfondo bianco. Il resto dell'applicazione condivide completamente le *view* tra i diversi *OS*. Per esempio la pagina di dettaglio dell'impianto è sempre un *layout* a *tab*. Per le liste che richiedevano immagini e colori descrittivi ho deciso di utilizzare i cerchi, questo per rendere più delicato l'impatto alla vista. Ho infine fatto largo uso dei bottoni in alto a destra nella barra di navigazione perchè molto diffusi in altre applicazioni. Essi inoltre consentono di risparmiare spazio per altro contenuto all'interno dei vari *layout*.

#### 3.5.2.4 Lista impianti

Dopo aver effettuato l'accesso l'utente si trova di fronte a una lista di impianti. In caso di *account* di tipo cliente vengono visualizzati gli impianti ad esso as-

sociati, in caso di un *account* tecnico vengono visualizzati tutti gli impianti sui quali esso ha almeno i privilegi di lettura. In caso la lista contenga più di 5 elementi l'utente visualizza anche una *box* di ricerca, per nome, codice o indirizzo. Solo un *account* tecnico può visualizzare dati riservati come il tipo di contratto.

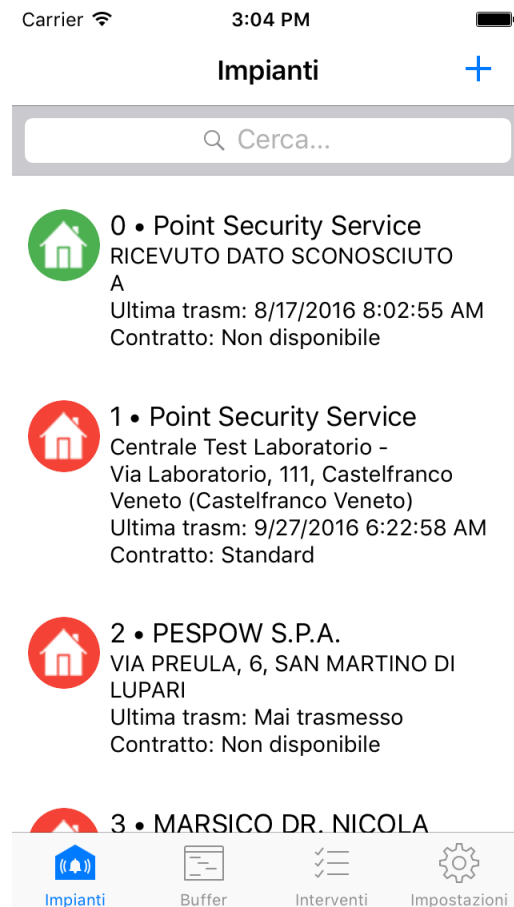


Figura 31: *Screenshot* della pagina contenente la lista di impianti visualizzabili da un tecnico (iOS).

### 3.5.2.5 Dettagli impianto

Una volta selezionato un impianto, l'applicazione scarica tutti i dati specifici ad esso collegati. La prima *tab* mostra lo storico degli eventi inviati dall'impianto, filtrabili per data. Le altre *tab* contengono rispettivamente: la lista di gruppi, la lista di sensori, la lista di utenti e la lista di comandi. Nella parte in alto a destra vengono aggiunti due bottoni differenti in base all'accesso: uno per la

creazione di un nuovo *ticket* in caso di accesso come cliente e uno per visualizzare le informazioni avanzate in caso di accesso come tecnico.



Figura 32: *Screenshot* della pagina di dettaglio dell'impianto (Android).

### 3.5.2.6 Comandi

Alcuni impianti accettano comandi remoti tramite protocollo IP. L'applicazione scarica quelli disponibili associando un colore e un'immagine a quelli più comuni. Ho utilizzato questa specifica *view* ricorsivamente in quanto visualizza sia comandi diretti, come "Inserimento totale", sia categorie di comandi, come "Gruppi", sia categorie di categorie. Quando un comando diretto viene selezionato l'applicazione chiede il codice di sicurezza e invia poi la richiesta. Quando invece l'utente preme su una categoria, riutilizzo la *view* per visualizzare le categorie o i comandi facenti parte alla categoria selezionata.



Figura 33: *Screenshot* della tab per comandare l'impianto (Windows).

### 3.5.2.7 Creazione *ticket*

Per la creazione di un nuovo *ticket* l'applicazione richiede semplicemente un nome di riferimento, un titolo per il riconoscimento immediato e una descrizione più approfondita del problema. Premendo il bottone in alto a destra si invia la richiesta al *server*.

Apri un ticket

Il tuo nome

Mario Rossi

Titolo

Batteria scarsa

Tipo ticket

Assistenza Tecnica

Descrivi la problematica

Ho bisogno di una nuova batteria

Figura 34: *Screenshot* della pagina di creazione di un nuovo *ticket* (Windows).

#### 3.5.2.8 Impostazioni

Attualmente non ci sono molte impostazioni per l'applicazione. Oltre allo spegnimento del *GPS* per gli *account* tecnico, sono presenti solo i filtri per lo storico degli eventi. Ogni evento appartiene a una categoria e non sempre l'utente vuole visualizzare lo storico completo per motivi di velocità e chiarezza. Quindi ho aggiunto dei *checkbox* per selezionare le categorie da visualizzare nella schermata di dettaglio dell'impianto.

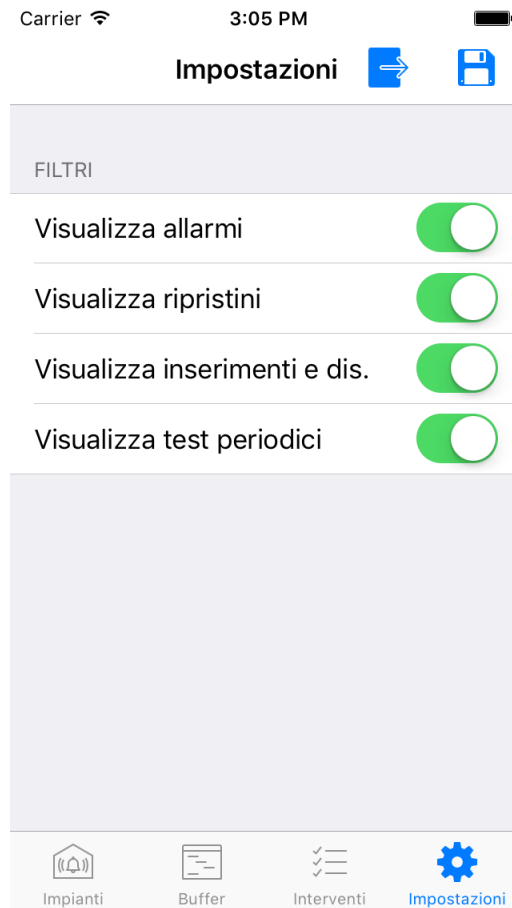


Figura 35: *Screenshot* della pagina delle impostazioni (iOS).

### 3.5.3 Funzioni avanzate

Le funzioni avanzate sono descritte nella tabella dei requisiti come desiderabili. Nella sezione seguente descriverò il lavoro da me portato a termine per soddisfare questi requisiti.

#### 3.5.3.1 Chat *ticket*

La realizzazione di una *chat* è stata una novità per me quindi ho dovuto pensare ad un modo per modificare l'estetica di un messaggio in base al mittente rispettando il pattern MVVM. Ogni messaggio ha una proprietà booleana che indica se il mittente sia l'utente o l'azienda, ho creato diversi *converter* per convertire il booleano in tipi adatti a modificare le seguenti proprietà grafiche:

- ***HorizontalAlign***: Il testo dei due mittenti si trova in lati opposti;

- **Margin:** I messaggi dell'utente sono distanti dal lato sinistro, quelli aziendali dal lato destro;
- **Color:** Mittenti diversi hanno colori diversi.

Grazie all'uso dei *converter* si possono effettuare *binding* tra proprietà di diverso tipo. Questo tipo di oggetto è efficace ed è stato quindi utilizzato molte volte in tutta l'applicazione, tanto da superare la dozzina di *converter* differenti.

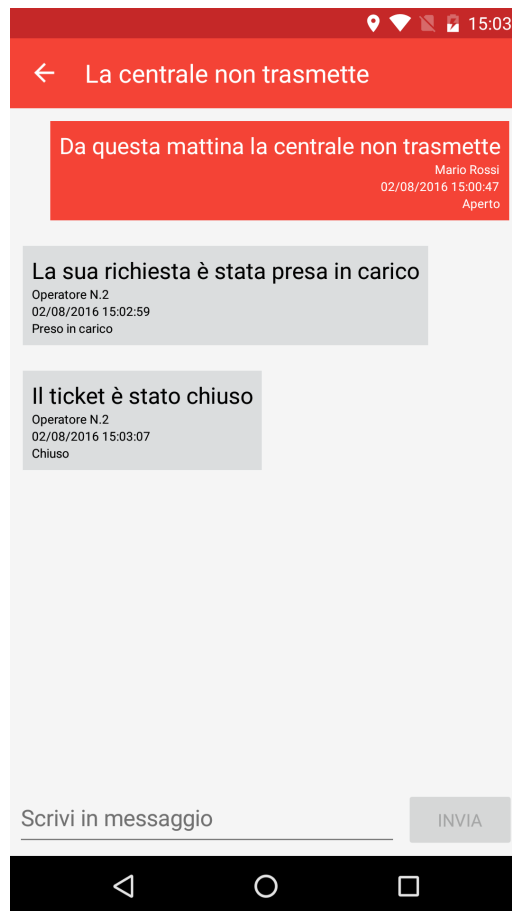


Figura 36: *Screenshot della chat di un ticket (Android).*

### 3.5.3.2 Gestione interventi

Ad ogni tecnico possono essere assegnati degli interventi da svolgere durante la giornata. Del menù principale il tecnico può accedere agli interventi a lui assegnati, selezionandone uno ne vede i dettagli. Dalla pagina di dettaglio è possibile gestire l'intervento in vari modi: prenotandolo, iniziandolo e finendolo.



Questo permette di velocizzare il lavoro d'ufficio per la fatturazione, per esempio calcolando le ore di lavoro in modo automatico.

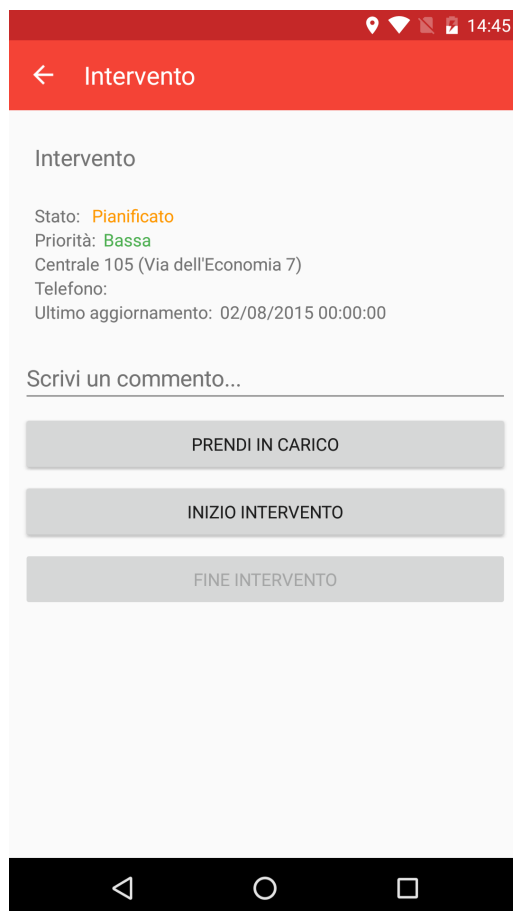


Figura 37: *Screenshot* della pagina per la gestione di un intervento da parte di un tecnico (Android).

### 3.5.3.3 Geolocalizzazione

La gestione della localizzazione non è stata sviluppata in Xamarin.Formsdata la grande differenza tra i sistemi operativi. Di conseguenza, per utilizzare questi servizi su tutte le piattaforme, abbiamo creato delle classi separate nei vari progetti nativi. Dopo l'accesso, ogni piattaforma chiama le proprie *API* native di geolocalizzazione le quali, utilizzando la PCL per la comunicazione con le *API web*, invia periodicamente la posizione dello *smartphone*. La funzione di localizzazione è attiva solamente per gli *account* di tipo tecnico, conoscendo la

posizione di ogni tecnico l'azienda può inviare fisicamente la persona più vicina ad un problema imprevisto.

#### 3.5.3.4 Chiamate e invio *email*

L'apertura di applicazioni esterne è una funzione avanzata e diversa per ogni piattaforma. Anche qui ho dovuto quindi creare classi diverse in ogni progetto nativo. Tutte le classi implementano la stessa interfaccia, la quale viene utilizzata dalle classi del progetto condiviso per eseguire le operazioni. Per inizializzare classi diverse di una stessa interfaccia in base alla piattaforma di esecuzione, viene usata la *reflection*. Questa tecnologia permette di creare istanze a tempo di esecuzione senza conoscerne il tipo dell'oggetto in fase di compilazione.

#### 3.5.3.5 Notifiche *push*

Le notifiche *push* sono una parte fondamentale di un'applicazione *mobile*. Anche in questo caso le librerie di Xamarin.Forms non permettono la gestione tramite un sistema unificato data la grande differenza tra le tre piattaforme di notifica. Android utilizza il Google Cloud Messaging e vanno importate librerie esterne create per Xamarin.Android nel progetto nativo, iOS e Windows invece utilizzano le librerie contenute nei rispettivi *SDK*. In ognuno dei casi è stato comunque necessario creare delle classi appostite nei progetti nativi per utilizzare questi servizi. Questa parte dello sviluppo è stata probabilmente la più dura, ogni sistema operativo ha *server* differenti ai quali registrarsi e mandare le informazioni, inoltre queste ultime hanno formato e contenuto diversi.

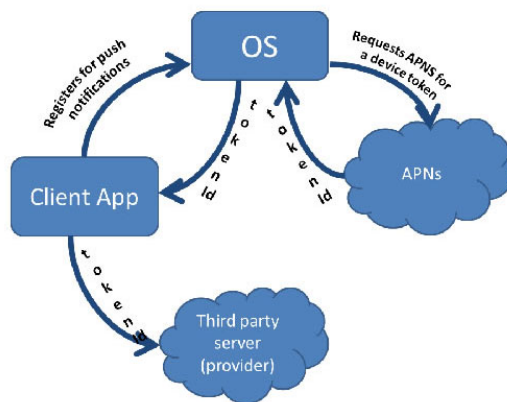


Figura 38: Schema di esempio per le notifiche *push* di iOS.

## 4 Testing

### 4.1 Analisi statica

L'analisi statica è un processo di verifica del codice sorgente che avviene senza l'esecuzione del *software*. Lo strumento utilizzato per l'analisi statica è l'ambiente di sviluppo Visual Studio. Esso mette a disposizione un sistema di analisi statica che visualizza dei *warning* in caso di codice possibilmente errato, ad esempio cicli infiniti o istruzioni condizionali che restituiscono sempre lo stesso risultato.

Inoltre è stata installata l'estensione Resharper per Visual Studio, essa aggiunge e migliora molte funzioni dell'*IDE* ed è una delle più installate al mondo. Una delle funzioni più utili è l'ottimizzazione del codice, suggerisce ed eventualmente riscrive le parti di codice non ottimale per aumentarne le prestazioni.

### 4.2 Analisi dinamica

L'analisi statica è il processo di verifica tramite l'esecuzione del *software*. Dato il ridotto tempo a disposizione non sono stati previsti test automatici, ogni prova è stata fatta manualmente.

Gli strumenti più utilizzati per il testing sono stati gli emulatori, essi permettendo un *deploy* veloce e in circostanze controllate, rendendo i test sempre ripetibili. Ho utilizzato diversi tipi di emulatori, soprattutto per il testing delle *view* con schermi di dimensione e risoluzione diversi.

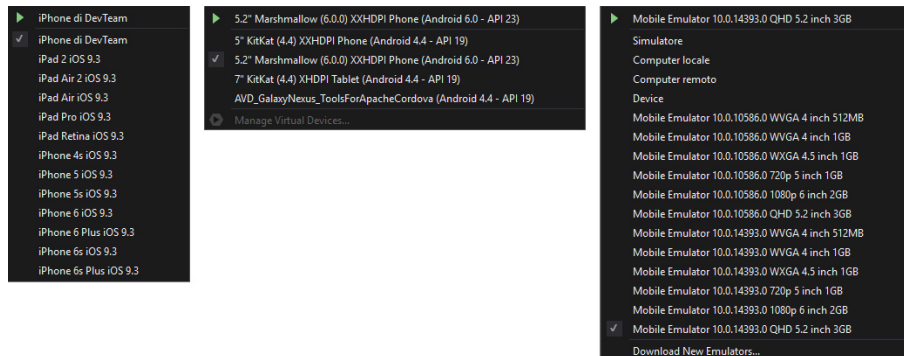


Figura 39: Lista degli emulatori disponibili durante lo sviluppo.

Alcune funzioni avanzate come la geolocalizzazione e le notifiche *push* sono molto difficili da testare con gli emulatori, per questo motivo ho eseguito e provato le applicazioni anche su svariati dispositivi reali. Ogni funzione rappresentante un requisito è stata testata sia sugli emulatori che sui dispositivi reali di ogni

piattaforma.

Per permettere il test anche a persone esterne allo sviluppo, il *tutor* ha creato dei canali di *beta testing* negli store *Android* e *Windows* aggiungendo dei *tester* alle *mailing list* dei canali. Questo permette di rilasciare l'applicazione in *beta*, facendola così provare solo alle persone selezionate. Ogni qual volta sviluppavo e testavo personalmente un funzione, rilasciavo l'applicazione aggiornata nel canale *beta*. Questo ha permesso di ricevere *feedback* continui, risparmiando così tempo di test a favore dello sviluppo.

Per mancanza di tempo non ho creato batterie di test da fare l'analisi dinamica automatizzata. Sicuramente l'esecuzione di test di unità automatici, grazie al *server CI*, avrebbero portato a un prodotto possibilmente più stabile e sicuro ma il lavoro dei *beta tester* ha parzialmente risolto questo problema.

## 5 Conclusione

### 5.1 Obiettivi raggiunti

#### 5.1.1 Imposti dall'azienda

Ho soddisfatto tutti i requisiti necessari, identificati dunque con una «**n**»:

- Ho imparato ad utilizzare al meglio l'ambiente di sviluppo Visual Studio comprese le varie componenti ed estensioni;
- Ho eseguito, insieme al *tutor* aziendale, l'analisi dei requisiti, dividendoli in basilari e supplementari;
- Ho studiato Xamarin.Forms prendendo familiarità con questa tecnologia e dei suoi linguaggi;
- Ho imparato ad utilizzare le *API web* per la lettura e scrittura di dati;
- Ho implementato tutte le *view* per le funzioni basilari;
- Ho sviluppato le funzionalità base;
- Ho concluso l'intera applicazione;
- Ho steso dei resoconti settimanali utilizzati durante le riunioni.

Ho svolto inoltre tutte le attività per soddisfare i requisiti desiderabili, identificati con una «**d**»:

- Ho acquisito un'ottima familiarità con le dinamiche di *team*;
- Ho compreso ed utilizzato tecniche di *debug* e test;
- Ho implementato tutte le *view* per le funzionalità supplementari;
- Ho sviluppato le funzionalità avanzate;
- Ho integrato tutte le funzionalità supplementari nell'applicazione finita.

Per quanto riguarda i requisiti opzionali, identificati con una «**o**», ho soddisfatto un unico requisito. Ogni qual volta trovavo una soluzione a un problema non comune o scoprivo metodi più efficaci per raggiungere un obiettivo l'ho riportato al *tutor* aziendale. Non sono riuscito invece a proporre l'aggiunta di funzioni non previste, ho però proposto e realizzato molte volte miglioramenti minori.

A *stage* concluso, ho quindi soddisfatto 14 requisiti su un totale di 16 previsti dall'azienda: 8/8 necessari, 5/5 desiderabili e 1/3 opzionali.

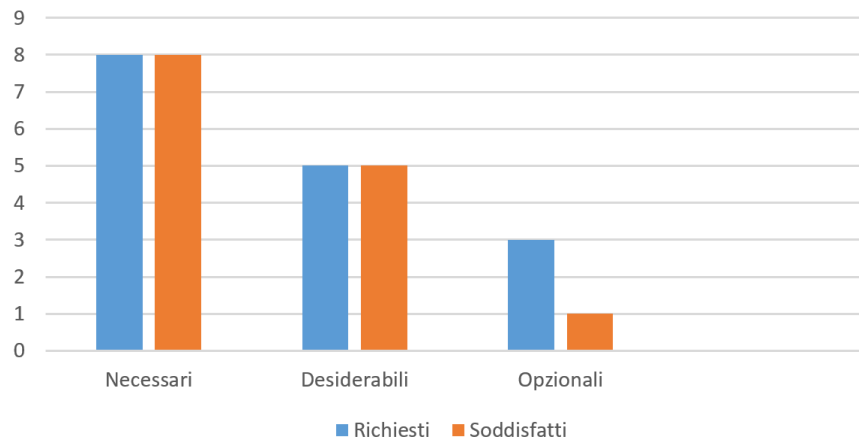


Figura 40: Grafico copertura requisiti.

### 5.1.2 Propri del progetto

La prima macro attività da svolgere durante lo *stage* è stata l'analisi dei requisiti, durante la quale io e il *tutor* ne abbiamo individuati 45. Al termine dal lavoro, dopo il collaudo, tutti i requisiti sono risultati soddisfatti.

### 5.1.3 Personali

Lo *stage* è stato un'opportunità e un'esperienza ottima per la mia crescita e formazione professionale. Durante le 8 settimane ho raggiunto gli obiettivi che mi ero prefissato riguardo lo sviluppo *mobile*, mi sono avvicinato a questo mondo come speravo, tanto da sentirmi autonomo nello sviluppo di un'eventuale applicazione futura.

Uno dei miei obiettivi era comprendere meglio le meccaniche di lavoro di *team* all'interno in un'azienda. Collaborando spesso con il *tutor* e utilizzando al meglio delle mie capacità gli strumenti offerti, ritengo di averlo raggiunto con successo.

## 5.2 Difficoltà affrontate

Dopo aver utilizzato per due mesi Xamarin.Forms ho imparato ad apprezzarlo e la ritengo una tecnologia ottima tanto da voler riutilizzarla in futuro. Purtroppo anche questo tipo di sviluppo non è esente da difetti. Le difficoltà più grandi si riscontrano quando vanno utilizzate le funzioni native di ogni piattaforma, come ad esempio le notifiche. Dover studiare tre approcci diversi per raggiungere lo stesso obiettivo richiede molto tempo e risulta frustrante, soprattutto se si sta utilizzando una tecnologia pensata per condividere la maggior quantità possibile

di codice.

Un altro problema che mi ha procurato diverse difficoltà è stato il comportamento inaspettato di alcune parti di codice su piattaforme specifiche. Spesso, utilizzando funzioni specifiche, mi è capitato di riscontrare comportamenti anomali dell'applicazione quando eseguita su una particolare piattaforma. Questi avvenimenti hanno rallentato lo sviluppo in quanto è stato difficile capire la fonte del problema. Inoltre la natura *wrapper* di Xamarin.Forms non permette un *debugging* efficace degli errori per le funzioni a basso livello e questo ha reso la risoluzione dei problemi di questo tipo molto più lenta.

### 5.3 Bilancio formativo

Questo *stage* ha arricchito le mie conoscenze su tutti i fronti. L'aspetto più importante è sicuramente l'aver affinato le mie capacità di lavoro in *team* in quanto posso essere migliorato solamente con l'esperienza. Se il progetto avesse coinvolto più persone ne avrei giovato, ma mi ritengo comunque più che soddisfatto.

Ho iniziato lo *stage* conoscendo già l'ambiente di sviluppo e i linguaggi utilizzati, conoscere però Xamarin.Forms è stato molto interessante, è una tecnologia nuova e con un potenziale molto alto, che se sviluppata può raggiungere anche altre piattaforme. Ritengo che per progetti come questo, nei quali non devono essere usate funzionalità specifiche dei sistemi operativi, Xamarin.Forms sia lo strumento perfetto e lo terrò sicuramente in considerazione per altri progetti.

Al di là della tecnologia utilizzata, sviluppare per *smartphone* e *tablet* costringe a progettare anche pensando all'usabilità e alle prestazioni del *software*. Credo che un'esperienza come questa sia necessaria per comprendere al meglio queste esigenze specifiche.

Il corso di studi che secondo me si è rivelato più importante durante questa esperienza è Ingegneria del *Software*, grazie all'esperienza data dal progetto didattico ho potuto effettuare l'analisi dei requisiti in modo veloce ed efficace. Inoltre, il progetto scolastico mi ha preparato al lavoro di *team* e all'utilizzo di tecnologie come il controllo del codice sorgente. Purtroppo il corso non ha approfondito la metodologia Scrum e l'ideologia *agile* in generale, per questo motivo ho dovuto studiare il metodo di lavoro completamente, azione che ha sottratto tempo allo sviluppo. In generale, il percorso di studi triennale mi ha preparato sufficientemente per poter affrontare questo *stage*, ma ritengo l'assenza di un corso sulle tecnologie *mobile* e il poco approfondimento di Scrum due mancanze considerevoli.