Alma Mater Studiorum · Università di Bologna

CAMPUS DI CESENA
SCUOLA DI SCIENZE
Corso di Laurea in Ingegneria e Scienze Informatiche

Un sistema di controllo accessi basato su tecnologie wearable

Relazione finale in
Programmazione di Sistemi Embedded

Relatore:
Chiar.mo Prof.
Alessandro Ricci

Co-relatore:
Lorenzo Francioni

Presentata da:
Nicola Giancecchi

Sessione II
Anno Accademico 2014/2015

Parole chiave

Wearable
Internet-of-Things
Controllo accessi
Bluetooth
Apple Watch

"You can't connect the dots looking forward; you can only connect them looking backwards. So you have to trust that the dots will somehow connect in your future. You have to trust in something - your gut, destiny, life, karma, whatever. Because believing that the dots will connect down the road will give you the confidence to follow your heart even when it leads you off the well-worn path and that will make all the difference."

Steve Jobs

Introduzione

Negli ultimi anni il mondo ha assistito ad un'espansione esponenziale del mercato mobile, un settore che ha rivoluzionato le nostre vite e le nostre abitudini. Parallelamente, sono entrati in commercio i primi dispositivi commerciali dell'era dell'Internet delle Cose (*Internet-of-Things*), un paradigma secondo il quale oggetti utilizzati quotidianamente possono dialogare tra loro mediante Internet al fine di migliorare la vita personale e i processi produttivi aziendali.

Un'applicazione pratica di *Internet-of-Things* sono i dispositivi indossabili (*wearable*) capaci per mezzo di sensori di raccogliere dati sanitari, ambientali e fisici attorno a noi.

I dispositivi indossabili attualmente in commercio permettono di collegarsi tramite protocolli e standard come WiFi e Bluetooth ad altri dispositivi, siano essi indossabili, smartphone o computer.

Da questo concetto nasce l'idea di far comunicare un indossabile con altri dispositivi utilizzati in un contesto aziendale. In questo documento verrà preso in considerazione un tassello fondamentale nella gestione delle risorse umane e della sicurezza: il controllo degli accessi in un edificio e lo sblocco di un varco, sia esso un tornello, una porta o una serranda. Verrà quindi analizzato e progettato un sistema che possa effettuare le operazioni di timbratura e di apertura di varchi dall'orologio Apple Watch e da iPhone collegandosi ad un terminale dedicato.

Il progetto nasce da un desiderio di potenziare e velocizzare un task quotidiano effettuato dagli operai o dagli impiegati di un'azienda quale è il controllo degli accessi. Molte volte può capitare di smarrire il portafoglio o il badge o di lasciarlo altrove mentre serve per la timbratura. Negli stabilimenti delle grandi aziende questo inconveniente si traduce in una perdita di tempo. Inoltre, come constatato negli incontri di preparazione della tesi, esistono

numerose tecnologie e standard per i badge, le quali non sono compatibili tra loro e possono portare un dipendente ad avere badge diversi per diversi compiti, il che è decisamente scomodo.

Si è pensato quindi di dematerializzare il badge e renderlo un oggetto "cloud" accessibile da tutti i dispositivi compatibili con Bluetooth, siano essi i "classici" smartphone e tablet oppure dispositivi indossabili come smartwatch, smartband o smartglasses. Verrà comunque mantenuta la retrocompatibilità con il classico badge formato tessera.

Questo progetto di tesi nasce come collaborazione di ricerca e sviluppo tra due aziende, Mr. APPs s.r.l. di San Marino - per la quale lavoro e che ha fornito Apple Watch di test e supporto - e Zeitgroup s.a.s. di Rimini - la quale ha fornito gli strumenti hardware di controllo degli accessi e le rispettive competenze.

In questo documento di tesi saranno introdotti alcuni concetti chiave del mondo *IoT* e *wearable*; verranno quindi descritti gli strumenti utilizzati nello sviluppo e trattata l'analisi e la progettazione in dettaglio del prototipo del sistema.

In particolare, il presente documento di tesi è strutturato come segue:

Capitolo 1: verrà fornita una panoramica sulle tematiche dell'*Internet-of-Things* e dei dispositivi *wearable*. In particolare verrà analizzato il passaggio dal mercato mobile a quello *wearable*, l'inserimento di tali oggetti in un contesto *IoT* e di ubiquitous computing. Verrà approfondito quindi il tema dell'utilizzo di dispositivi *wearable* nel settore domotico.

Capitolo 2: in questo capitolo verrà presentata l'idea di base del progetto e il software e l'hardware utilizzato per la realizzazione del prototipo. Seguirà quindi una panoramica tecnica sui singoli componenti hardware, il software, i protocolli e gli standard adottati.

Capitolo 3: verranno analizzati i requisiti del sistema, i casi d'uso e le interazioni tra i protagonisti. Saranno analizzate le problematiche imposte dal progetto e verrà mostrata l'interfaccia grafica dell'applicazione.

Capitolo 4: verranno presentati i linguaggi e gli strumenti utilizzati per la progettazione e le soluzioni adottate. Saranno quindi mostrati i diagrammi UML per il modello, le viste e i controller utilizzati, realizzati in

un'ottica di progettazione MVC (*Model-View-Controller*). Scendendo nei particolari, si analizzerà l'architettura del sistema e i tipi di comunicazione adottati nei singoli scenari, con l'aiuto di schemi e listati di codice.

Capitolo 5: fornirà una valutazione generale sul progetto, i test effettuati e idee su come il sistema possa essere ampliato e migliorato.

Indice

Capitolo 1

Stato dell'Arte

1.1 Scenario: da mobile a wearable

Dal 2007, anno di presentazione dei primi smartphone, il mondo ha assistito a un cambiamento tecnologico nella vita quotidiana che è diventato sempre più radicale ed evidente. Due sono i principali fattori che hanno condizionato il cambiamento: l'evoluzione dell'hardware da una parte, con touchscreen capacitivi, componenti più potenti e maggiori requisiti di memoria, e il software dall'altro, con la diffusione di sistemi operativi mobile basati sulle controparti desktop: iPhone con Mac OS X, Android con Linux e, più tardi, Windows Phone con Windows.

Nel corso di questi anni il mercato mobile è cresciuto esponenzialmente: dai 172,38 milioni di smartphone venduti nel 2009, dove in testa c'era ancora Nokia con il suo Symbian, si è passati al 2014 con 1,25 miliardi di smartphone venduti, dei quali 1 miliardo è equipaggiato con Android, seguito a ruota da iOS e Windows Phone. [9]

Un grande motore di sviluppo economico e lavorativo è l'ecosistema delle app e dei relativi market online che genera introiti dalla vendita e dalle pubblicità: prendendo ad esempio i numeri di Apple App Store troviamo una media di 119 app scaricate per persona, 850 app scaricate al secondo per un totale di 100 miliardi di download, con 30 miliardi di dollari pagati agli sviluppatori di 1.5 milioni di app. [7]

In parallelo ultimamente si sono intravisti sul mercato i primi dispositivi commerciali indossabili (*wearable*), ovvero apparati tecnologici realizzati per essere indossati in maniera continuativa. Ogni dispositivo *wearable* può avere

Figura 1.1: Alcuni esempi di dispositivi *wearable* attualmente in commercio. Da sinistra in alto, in senso orario: Google Glass, Fitbit, fascia cardiaca Bluetooth Polar, Apple Watch

caratteristiche molto differenti dagli altri. Possiamo, in linea di massima, raggrupparli in due macrocategorie:

- monotasking, ovvero incentrati su un unico task da eseguire - principalmente il rilevamento di parametri vitali, fisici o ambientali. Fanno parte di questa famiglia gli activity tracker (Nike+), le smartband (Fitbit) e i capi di moda che includono microcomputer;

- multitasking, capaci di eseguire più funzionalità insieme. Questi dispositivi possono integrare le funzionalità di uno smartphone e uniscono in un solo dispositivo più indossabili monotasking. Appartengono a questa categoria gli smartwatch (Apple Watch, Samsung Gear, Moto360), gli smart glasses (Google Glass, Vuzix, Moverio) e i visori di realtà aumentata (Oculus Rift, Microsoft HoloLens).

La figura 1.1 mostra alcuni esempi di dispositivi *wearable* attualmente in commercio.

Questi dispositivi si interfacciano e supportano la maggior parte degli smartphone grazie a tecnologie open source o standard come Bluetooth e WiFi. Essi sono caratterizzati da un'interazione completamente rivisitata rispetto a quella che già conosciamo con gli smartphone: un dispositivo *wearable* è pensato per eseguire task di breve durata, quindi anche l'interfaccia e le app dovranno essere progettate tenendone conto.

Nonostante questo aspetto però, tipicamente un dispositivo *wearable* lavora continuamente in background per raccogliere dati, siano essi vitali, fisici o ambientali. Ciò significa che il software e l'hardware devono essere ottimizzati per ridurre al minimo l'utilizzo della batteria.

Una classe di dispositivi degni di nota sono gli indossabili in fase di sperimentazione nell'ambito medico dedicati al monitoraggio di parametri vitali come sangue, battiti cardiaci (elettrocardiogramma), attività muscolare, elettroencefalogramma e glucosio - anche tramite controllo remoto. Le ricerche in questo ambito stanno spingendo verso la realizzazione di tecnologie semipermanenti sul corpo, siano esse l'impianto di "tatuaggi" sulla pelle (permanenti e non) dotati di chip e sensori o l'applicazione di piccoli dispositivi sottocutanei [3].

1.2 Internet-of-Things

Tali dispositivi si inseriscono in un contesto più ampio, oggetto di studi accademici e commerciali. Si introduce quindi il concetto di *Internet-of-Things* (*IoT*), in italiano l'"Internet delle Cose", un termine coniato nel 1999 dall'imprenditore inglese Kevin Ashton nell'ambito delle catene di montaggio [1].

Per *Internet-of-Things* si intende un insieme di oggetti reali che integrano tecnologie embedded - come circuiti RFID (*Radio-Frequency Identification*), sensori, attuatori, circuiti elettronici, etc. - in grado di comunicare e cooperare con altri dispositivi inseriti in una rete personale (PAN, *Personal Area Network*) o espansa (WAN, *Wide Area Network*). Tali oggetti prendono nome di *smart objects*.

Lo scopo principale di un contesto *IoT* è quello di assistere e aiutare l'utente finale nella vita di ogni giorno tramite smart objects di vario tipo, interconnessi ed inseriti in ambienti domestici, lavorativi e urbani. Tra i più comuni troviamo i dispositivi per la salute, la domotica e la didattica. *IoT*

svolge un importante ruolo anche nelle applicazioni industriali, logistiche, di trasporto e business [2].

Nel 2008 il numero di "cose" connesse ad Internet (compresi quindi computer e smartphone) ha superato il numero di persone in vita sulla Terra. Si stima perciò che entro il 2020 i dispositivi connessi ad Internet saranno circa 50 miliardi, vale a dire 7 volte la popolazione mondiale [3].

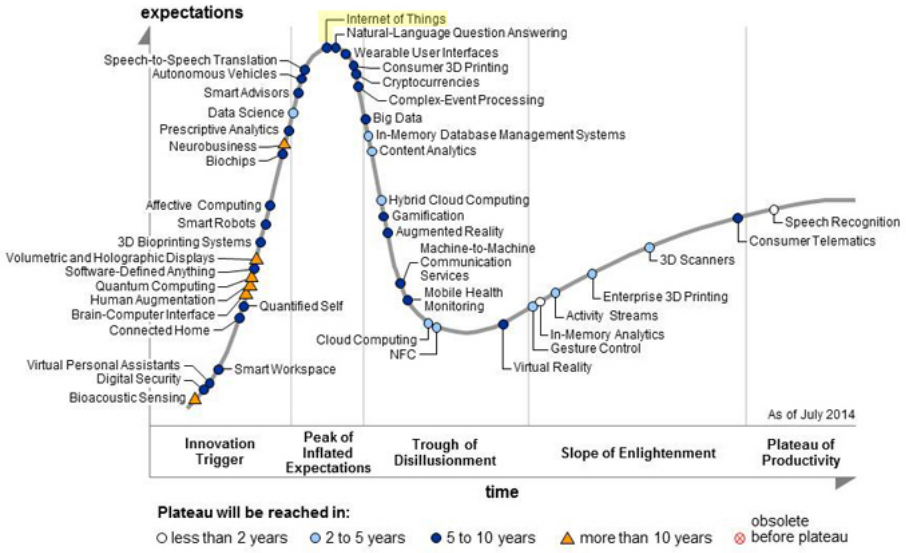


Figura 1.2: Gartner, Inc., *Gartner's 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business*

Una ricerca condotta da Gartner nel 2014 [10] individua *IoT* come l'asset con maggiori aspettative che raggiungerà il mercato consumer in maniera massiva entro 5-10 anni (grafico in fig. 1.2).

1.2.1 Ubiquitous Computing (ubicomp)

IoT fa parte di una tematica teorica più ampia, ovvero l'*ubiquitous computing* (abbreviato ubicomp), che in italiano potremmo tradurre come "informatica onnipresente", la quale nasce una decina di anni prima dell'avvento di *IoT*. Mark Weiser, uno dei primi ricercatori in materia arruolato allo Xerox Palo Alto Research Center (PARC) negli anni '80, definisce l'ubicomp come un nuovo campo dell'informatica basato su un mondo fisico ricco di sensori, attuatori, schermi ed elementi computazionali, invisibilmente collegati tra

loro per mezzo di una rete e perfettamente integrati nella vita quotidiana [4]. Non solo: l'avvento dell'ubicomp può portare miglioramenti anche e soprattutto in campo industriale.

Le teorie visionarie del PARC iniziano a prendere forma in primis con l'avvento di Internet, ma soprattutto con l'arrivo verso la fine degli anni 2000 del *Cloud Computing*. Con cloud si intende l'erogazione di risorse informatiche (hardware, storage, software) a richiesta dell'utente tramite la rete Internet. Commercialmente parlando, tali risorse diventano "servizi": si possono utilizzare, ampliare o ridurre risorse (scalare) praticamente all'infinito in modalità completamente trasparente all'utente, poichè queste risorse risiedono su una "nuvola", ovvero cluster di server ridondati sparsi in giro per il mondo.

Il Cloud Computing si è sviluppato in maniera esponenziale sia grazie agli sforzi di gestori privati, ma soprattutto grazie agli investimenti dei principali colossi informatici per la costruzione di infrastrutture cloud pubbliche quali Amazon (Amazon Web Services), Microsoft (Windows Azure), Google (Google Cloud) e Apple (iCloud, il quale si appoggia anche ad Azure e AWS). Basti pensare che molte startup di successo come Dropbox, Netflix, Airbnb o Foursquare sono state sviluppate sull'infrastruttura di Amazon Web Services per via della sua scalabilità e dei costi di mantenimento, che rimangono comunque minori rispetto alla costruzione di un'infrastruttura propria ridondata.

L'architettura cloud e le sue caratteristiche sono le fondamenta dell'IoT, proprio per il fatto che i dispositivi producono una gran mole di dati - i *big data*, che richiedono una facile scalabilità - e che la maggior parte delle aziende non riesce a permettersi infrastrutture proprie.

1.3 Wearable e domotica: possibili sviluppi

Un argomento caldo degli ultimi anni sul quale sono state investite molte risorse è la domotica (*home automation*), ovvero l'applicazione di tecnologie elettroniche ed informatiche per il controllo - anche remoto - di abitazioni e ambienti lavorativi volta al miglioramento della qualità della vita.

Quello domotico è un settore non certamente nuovo: le prime applicazioni risalgono infatti agli anni sessanta dello scorso secolo. Tuttavia il progresso informatico degli ultimi decenni, l'arrivo di Internet e, in ultima battuta, gli studi sull'IoT hanno portato nuova linfa all'evoluzione di questo settore.

Molti colossi di questa industria hanno introdotto ultimamente apparecchiature di ultima generazione per il controllo domotico della casa dall'interno (grazie all'utilizzo di tablet creati ad hoc e montati a muro) e anche dall'esterno, tramite la rete Internet e il collegamento di applicazioni o pannelli web.

Sviluppatori e ingegneri di tutto il mondo si stanno interessando all'argomento, tanto da spingere le grandi aziende a venire incontro a questa domanda. Per citare un esempio, la italiana BTicino dal 2000 sviluppa un protocollo chiamato *OpenWebNet* per l'interazione dall'interno e dall'esterno con i propri sistemi.

Internet-of-Things ha spinto molti produttori a progettare componenti indipendenti già abilitati per la connessione alla rete WiFi casalinga senza bisogno di costruire nuovi impianti elettrici. Due dei casi più famosi in ambito commerciale sono la lampada connessa *Philips Hue*, la cui accensione, luminosità e colore sono comandabili tramite un'app e *Nest*, il termostato intelligente progettato da Tony Fadell, uno dei padri di iPod, e commercializzato da Google che ha riscosso un ottimo successo negli USA. Degno di nota è anche *HomeKit*, il framework domotico presentato da Apple durante la WWDC 2014 dedicato alla gestione dei dispositivi casalinghi di qualsiasi tipo (lampade, serrande, interruttori, etc...) connessi in rete.

Il passo successivo riguarderà certamente l'interazione con i nuovi dispositivi *wearable* in commercio e le sperimentazioni in atto oggetto di ricerche industriali e accademiche. Come già accennato, i *wearable* possiedono molti più sensori rispetto ad uno smartphone; se tali sensori venissero sfruttati dalla domotica produrrebbero un'ambiente domestico o lavorativo che si adatta automaticamente ai bisogni dell'utente.

Alcuni esempi pratici potrebbero essere:

- un termostato intelligente come *Nest* che possa controllare la nostra temperatura corporea tramite un dispositivo *wearable* ed alzare o abbassare la temperatura ambientale in base ad essa;
- una serratura elettronica che possa aprirsi all'avvicinamento dell'utente alla porta (previa verifica dell'identità o conferma da parte sua) e possa accendere luci o aprire serrande in base all'orario e alle abitudini;
- un'applicazione che tenga sotto controllo i battiti cardiaci e la frequenza di respirazione dell'utente per spegnere automaticamente le luci e chiudere le serrature in caso di addormentamento.

A livello accademico sono stati svolti alcuni esperimenti, tra i degni di nota si segnala un ciondolo indossabile dotato di infrarossi che permettono di catturare gesture associate a determinati eventi per attivare o disabilitare componenti domotici realizzato da T. Starner e altri [5].

Capitolo 2

Il progetto

2.1 L'idea

L'idea è quella di realizzare un sistema che consenta l'accesso ad un edificio o ad una stanza soltanto in presenza di un dispositivo associato all'utente che desidera entrare. Tale dispositivo, in linea teorica, può essere uno smart object qualsiasi con supporto a Bluetooth e Wi-Fi.

Questi dispositivi permettono di affiancare e sostituire il classico badge che viene fornito, ad esempio, alle risorse umane di un istituto, un'azienda o un'università.

I vantaggi di questo approccio sono molteplici:

Dematerializzazione: il badge non è più una tessera di plastica ma una credenziale d'accesso memorizzata su un server remoto. Ciò permette di avere il badge virtuale su più dispositivi di uso quotidiano, anzichè una sola copia nel proprio portafoglio.

Aggiornamento: una problematica riscontrata è quella delle diverse tecnologie di badge adottate dalle aziende, il che porta l'utente ad avere con sè anche più tessere contemporaneamente in caso di grandi stabilimenti. Il badge virtuale non presenta questa necessità: anzi, come verrà analizzato più avanti, sarà possibile rendere il badge sia retrocompatibile che compatibile con le nuove versioni dei sistemi.

Controllo immediato: l'accesso all'edificio/stanza può essere abilitato o revocato in qualsiasi momento da remoto dal proprietario del sistema, aspetto utile nel caso un utente cambi livello o mansione.

Il sistema quindi permetterà di:

- associare un varco ad uno o più utenti autorizzati all'accesso;
- accedere ad un varco, quindi sbloccare una serratura o tornello elettricamente;
- tenere traccia degli orari di entrata e uscita degli utenti da tale varco;
- avvisare l'utente dell'avvicinamento al varco.

Per facilitare l'accesso ed eliminare al massimo le complicazioni si è deciso di far compiere all'utente il minor numero di passaggi possibile, attraverso varie tecniche - salvataggio offline dei varchi autorizzati, notifiche locali, etc. - che verranno presentate più avanti.

Il presente progetto di tesi nasce come collaborazione di ricerca e sviluppo tra le aziende:

Mr. APPs s.r.l.: app agency per la quale lavoro con sede nella Repubblica di San Marino nata a novembre 2012. Mr. APPs progetta e sviluppa applicazioni mobile per sistemi iOS, Android e Windows. Realizza inoltre siti web, soluzioni e-commerce e app marketing. Ha all'attivo un portfolio in costante crescita di 33 app, 31 siti web e 14 e-commerce e una serie di traguardi e collaborazioni di alto livello con Apple, Google e Microsoft. Tutti i progetti sono realizzati nativamente utilizzando i framework e gli strumenti di sviluppo messi a disposizione dai produttori.

Zeitgroup s.a.s. : società con sede a Rimini dedita alla progettazione di sistemi per controllo accessi, rilevazione di presenze e controllo di produzione. Segue l'attività di vendita e post-vendita tramite progettazione, consegna, installazione e manutenzione dei terminali e dei sistemi software, basati anche su cloud.

Grande fonte d'ispirazione e di studio per l'ideazione di questa tesi sono state anche le Conferenze Mondiali degli Sviluppatori Apple (WWDC - *Apple Worldwide Developers Conference*) svoltesi a San Francisco (USA) alle quali ho avuto l'onore di partecipare nel 2014 e nel 2015 come vincitore di una borsa di studio messa in palio da Apple per gli studenti da tutto il mondo. Alcune tra le innovazioni e i framework analizzati e utilizzati in questa tesi sono stati presentati proprio durante queste conferenze, dove è stato possibile provarle in anteprima con l'aiuto degli ingegneri Apple.

2.2 Prototipo

Si è deciso quindi, sulla base dei requisiti del problema, di realizzare un prototipo del sistema in oggetto. Tale prototipo è in grado di leggere i badge virtuali associati a dispositivi Bluetooth 4.0 e di avvisare l'utente dell'esito dell'operazione nel momento in cui si avvicina al terminale.

Il prototipo si basa sui seguenti componenti:

Zucchetti Axess-TMC 930 X2 , terminale di controllo accessi e rilevazione presenze prodotto da Zucchetti Axess e fornito gentilmente da Zeitgroup;

Testina Bluetooth 4.0 e Beacon Estimote , connessi al rilevatore X2 ed utilizzati per la comunicazione con i dispositivi;

iPhone e Apple Watch utilizzati come dispositivi di sblocco;

Software realizzato per iPhone, Apple Watch e web server.

Prima di entrare nei dettagli analitici del progetto, vengono analizzati i componenti hardware e software utilizzati.

2.3 Hardware

2.3.1 Terminale di controllo accessi

Figura 2.1: Terminale 930 X2, vista frontale

È stato adottato un terminale modello *Zucchetti Axess-TMC 930 X2* per tutto ciò che riguarda il controllo degli accessi. Il terminale possiede di serie

un lettore RFID Clk&Data 125KHz MiFare ed è espandibile con altri lettori esterni, testine di lettura, lettori di impronte digitali o lettori generici con interfaccia Wiegand. In totale supporta 3 lettori, più altri due collegabili a schede di espansione NeoMAX. I lettori possono essere collegati tramite connettori molex o connettori a vite, in base alla tipologia di cavo. Per la gestione dei varchi di accesso viene fornito un relè interno che può commutare un carico massimo di 1 A a 30 V_{dc}.

Presenta inoltre una porta Ethernet RJ45 per il collegamento in rete, con supporto al Power-on-Ethernet (standard IEEE 802.3af), una batteria da 600 mAh, uno schermo LED monocromatico e un tastierino numerico per l'inserimento del PIN e di codici causali [8].

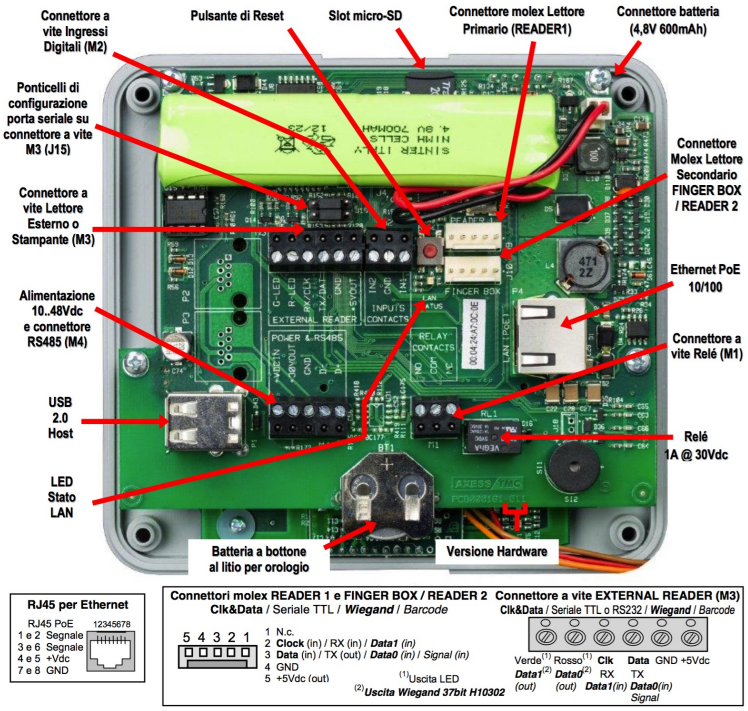


Figura 2.2: Terminale 930 X2, vista posteriore

Il sistema operativo del terminale è di tipo proprietario ed è ospitato su una scheda micro-SD estraibile, facilitando così l'aggiornamento del sistema e l'esportazione dei dati in modalità offline. Il sistema fornisce un pannello di controllo web Keil Embedded Web Server 2.0 prodotto da ARM e accessibile

in HTTP dall'indirizzo IP assegnato e visibile sullo schermo all'accensione del terminale.

Tramite tale pannello, o in alternativa tramite protocollo FTP, è possibile modificare i file utilizzati dal sistema per gestire gli accessi. Il "database" è infatti composto da file testuali con all'interno record di lunghezza fissa.

2.3.2 Bluetooth

Introduzione a Bluetooth

Come protocollo di comunicazione tra il terminale e il dispositivo di sblocco è stato impiegato Bluetooth 4.0 "Low Energy". La scelta è ricaduta su di esso per diversi motivi: un gran numero di dispositivi supportati, un consumo di corrente e risorse molto contenuto, un corto raggio di azione e un basso costo di produzione dei componenti.

La tecnologia Bluetooth è nata nel 1994 per iniziativa di un gruppo di ingegneri di Ericsson, in seguito affiancati anche da Intel, Nokia, Toshiba e IBM. Bluetooth opera sulle frequenze radio senza licenza ad uso industriale, scientifico e medico (ISM) da 2.4 a 2.485 GHz in modalità full duplex [11]. La banda è suddivisa in 79 canali di ampiezza 1 MHz, con un margine superiore di sicurezza di 3.5 MHz e uno inferiore di 2 MHz. Il protocollo fa uso del *frequency hopping*, una tecnica che permette di aumentare la larghezza di banda di un segnale commutando tra i canali 1.600 volte al secondo.

L'architettura di una rete Bluetooth è di tipo *master-slave* e si chiama *piconet*. Un dispositivo master può comunicare con 7 slave contemporaneamente, ma solo uno slave per volta può comunicare con esso. I dati vengono scambiati in sincronia con il clock del master che scatta ad intervalli di 312.5 µs. Più *piconet* collegate assieme formano una *scatternet*.

Ogni dispositivo Bluetooth, per connettersi ad un altro dispositivo, ha bisogno di sapere le tipologie di servizio offerte da quest'ultimo: tali servizi si chiamano "profili" e vengono utilizzati per uno scopo specifico. Tra questi ricordiamo a titolo esemplificativo *Headset Profile (HSP)* per auricolari e vivavoce, *Personal Area Network Profile (PAN)* per la creazione di reti Ethernet senza fili, *Advanced Audio Distribution Profile (A2DP)* per la fruizione di contenuti audio in stereofonia.

Nel 2010, dopo 4 anni di studi da parte di Nokia, viene introdotta la versione 4.0 di Bluetooth, conosciuta come "Bluetooth Low Energy" (in acronimo *BLE*) o, commercialmente parlando, "Bluetooth Smart". Nonostante non sia retrocompatibile con Bluetooth Classic, lavora sulle stesse frequenze ma con canali differenti: infatti *BLE* utilizza 40 canali da 2 MHz ciascuno. Come già accennato questo protocollo porta con sè molti vantaggi:

- minor utilizzo di energia;
- componenti più economici;
- quindi inclusione in molti più tipi di dispositivi - non più solo computer e smartphone.

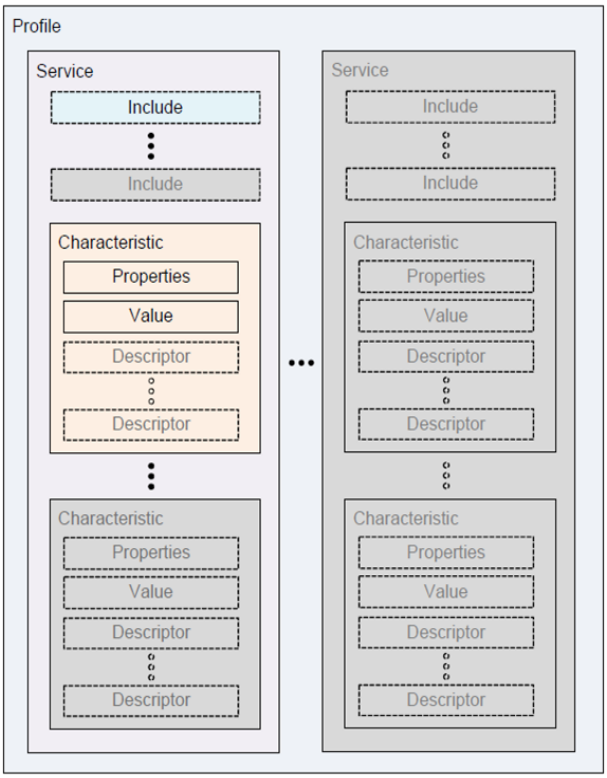


Figura 2.3: Architettura di un profilo GATT

Bluetooth 4.0 permette inoltre agli sviluppatori di fornire profili generici chiamati GATT (*Generic Attribute Profile*) per i client che si collegano ai

dispositivi (fig. 2.3). All'interno di un profilo GATT sono definiti diversi servizi, uno dei quali può essere quello che trasmette informazioni sul dispositivo come nome e identificativo. Ogni servizio fornisce dei parametri denominati "caratteristiche" (*characteristics*) composti da un valore e da descrittori. In base alle proprietà della caratteristica, i descrittori possono essere letti e/o modificati.

Testina Bluetooth

Al lettore Zucchetti Axess 930 X2 è stata collegata una testina compatibile con Bluetooth Low Energy. Tale testina comprende sulla propria scheda un microprocessore ARM Cortex M0+ a 32 bit modello NXP LPC824 e un chip Bluetooth 4.0 Bluegiga BLE113.

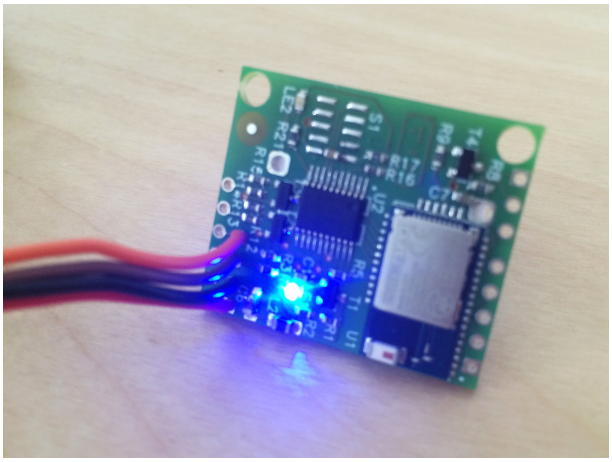


Figura 2.4: Testina Bluetooth 4.0 in funzione. Si notano il microprocessore NXP LPC824 (package nero sulla sinistra) e il chip Bluetooth Bluegiga BLE113 (grigio, sulla destra)

Il circuito lavora in Transistor-Transistor Logic (*TTL*), una tecnologia ampiamente utilizzata in realtà scientifiche ed industriali grazie ai suoi bassi costi di produzione.

La testina è compatibile anche con le versioni precedenti di Bluetooth, ampliando la base di potenziali utenti del sistema anche a persone che possiedono cellulari e smartphone antecedenti al 2011. Si ricorda infatti che il primo smartphone a supportare Bluetooth 4.0 è stato iPhone 4s e che ancora

molti utenti - soprattutto in ambienti come fabbriche o uffici - non possiedono smartphone di ultima generazione.

Estimote Beacon

La testina è stata affiancata da un beacon per poter avere informazioni sulla posizione del terminale. La testina infatti supporta Bluetooth Low Energy e, nonostante abbia caratteristiche tecniche simili ad un beacon, non è conforme agli standard *de facto* di Apple iBeacon e del recente Google Eddystone. Un'implementazione futura di questo progetto porterà all'unificazione dei due dispositivi in un'unica scheda.

Un beacon è un dispositivo che emette un segnale radio contenente un identificativo univoco (UUID) su onde radio Bluetooth. Grazie al profilo GATT di prossimità *PXP* viene trasmesso anche l'indicatore di potenza del segnale radio (RSSI) misurato in dB.

Figura 2.5: Un beacon Estimote

Il beacon utilizzato è prodotto da Estimote, una delle prime startup ad aver introdotto sul mercato questo tipo di dispositivi e ad aver creduto nel loro potenziale. Esso è equipaggiato con un chip ARM Cortex M0 a 32 bit, affiancato ad una memoria flash da 256 kB [12].

2.3.3 iPhone e Apple Watch

In questo progetto di tesi sono stati utilizzati iPhone e Apple Watch.

iPhone

iPhone è lo smartphone di Apple, Inc. presentato per la prima volta il 9 gennaio 2007 da Steve Jobs. Il primo modello, nonostante fosse carente di alcune funzioni, portò una grande rivoluzione nel settore degli smartphone grazie ad alcune caratteristiche di punta, tra le quali troviamo l'adozione di un display capacitivo, l'utilizzo di un sistema operativo derivato da un sistema desktop quale Mac OS X e la sottoscrizione di accordi speciali con i gestori americani per la fornitura di abbonamenti "bundle" con grandi quantità di dati. Questi sono solo alcuni dei motivi che hanno portato al successo di iPhone, al potenziamento di Android (esso infatti nasce nel 2003 e commercializzato solamente nel 2007) e alla diffusione massiva degli smartphone.

Durante questi otto anni sono usciti nuovi modelli che hanno apportato numerosi miglioramenti nell'hardware, nel design e nel software. Gli ultimi due modelli presentati, iPhone 6s e 6s Plus, sono caratterizzati da un processore proprietario Apple A9 a 1,51 GHz tri-core basato su ARMv8-A, 2 gigabyte di memoria RAM, display Retina ad alta definizione con 3D Touch - tecnologia già nota su Apple Watch come Force Touch, doppia fotocamera da 12 megapixel (posteriore) e 5 megapixel (frontale) con registrazione video in 4K.

Per il testing del progetto è stato utilizzato un iPhone 5s 16GB con sistema operativo iOS 8.4.1 e iOS 9.0.

Apple Watch

Apple Watch è il primo esperimento di smartwatch prodotto dalla casa di Cupertino. È stato presentato da Tim Cook il 9 settembre 2014 ed è stato messo in vendita dal 24 aprile 2015. La vastità di combinazioni possibili stona con la filosofia Apple ma non con quella del settore degli orologi: oltre a diversi tipi di cinturini, Apple Watch è disponibile in tre modelli (Sport, Watch e Edition) e in due dimensioni della cassa (38mm e 42mm) per un totale di 96 combinazioni.

L'orologio è basato su un processore RISC proprietario, Apple S1. Possiede un display Retina flessibile con supporto a Force Touch: grazie a questa tecnologia viene rilevato il livello di pressione sullo schermo, abilitando così nuove tipologie di interazione con l'interfaccia. Degno di nota anche il feed-

Figura 2.6: Apple Watch

back aptico e la corona digitale ad alta precisione. Watch può essere associato ad un iPhone per volta ed è compatibile da iPhone 5 in poi.

Apple Watch e iPhone 6 sono inoltre dotati di NFC (*Near Field Communication*) ma non sono abilitati a funzioni diverse dal pagamento con Apple Pay, il sistema di pagamenti contactless proprietario di Apple.

2.4 Software

Il software di questo progetto di tesi è stato sviluppato per i sistemi operativi iOS (iPhone) e watchOS (Apple Watch). È stato inoltre sviluppato un semplice webservice con il quale interfacciarsi per eseguire l'autenticazione.

2.4.1 iOS

I dispositivi mobile di Apple quali iPhone, iPad e iPod touch sono equipaggiati con iOS (precedentemente iPhone OS), un sistema operativo mobile basato su Mac OS X, quindi derivato da UNIX e con microkernel XNU Mach. I sistemi mobile Apple si basano su CPU proprietarie con architettura RISC (*Reduced Instruction Set Computer*) basate su ARM.

All'avvio, iOS presenta una schermata di lancio (*Springboard*) dove sono presenti le icone delle applicazioni installate, sia di sistema che non. Le app di sistema possono essere aggiornate solamente tramite un aggiornamento del sistema operativo, mentre per le app di sviluppatori terzi è presente l'App Store, il marketplace di riferimento per iOS.

iOS è un sistema chiuso e protetto: infatti non è possibile - se non tramite il *jailbreak* - installare applicazioni di terze parti non approvate da Apple e non presenti su App Store. Le applicazioni inoltre vengono eseguite in una *sandbox* e non possono accedere a componenti e risorse di sistema o di altre applicazioni se non tramite le API e gli SDK forniti da Apple.

2.4.2 watchOS e WatchKit

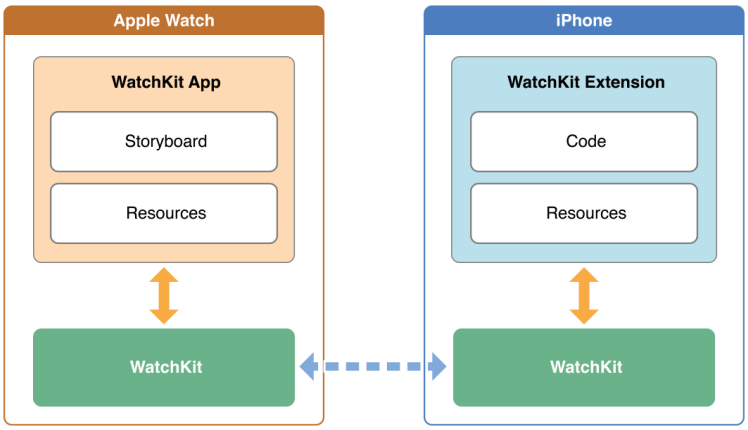


Figura 2.7: Architettura delle app di WatchKit 1.0

watchOS è il sistema operativo dell'orologio, basato su iOS. Permette l'installazione di applicazioni, la cui logica però rimane su iPhone, con il quale deve essere stabilito un collegamento continuo tramite Bluetooth e/o WiFi. In assenza di connettività alcune applicazioni di sistema rimangono disponibili offline, mentre il resto delle applicazioni non può essere eseguito.

Questo perchè WatchKit, il framework utilizzato per sviluppare app su Apple Watch, è suddiviso in due parti:

- la parte iPhone, dove è presente la WatchKit Extension che esegue il codice e dialoga con l'app iOS e le sue librerie;
- la parte Apple Watch, contenente solo le schermate e le risorse grafiche (WatchKit App).

watchOS 2, presentato alla WWDC 2015 e disponibile dal 16 settembre 2015, garantirà una minore dipendenza da iPhone grazie allo spostamento

della logica di funzionamento direttamente sull'orologio, oltre a nuove opportunità per gli sviluppatori come l'accesso agli SDK per complicazioni, corona digitale, Taptic Engine e cardiofrequenzimetro.

2.4.3 Tecnologie web

Per il corretto funzionamento del progetto è stato realizzato un semplice web service per le funzionalità di login e di download della lista di varchi di accesso autorizzati.

Il web service è basato su PHP ed utilizza il framework Symfony 2.7. Symfony è un framework PHP open source rilasciato sotto licenza MIT. Grazie ad un'architettura basata interamente su MVC, con Symfony è realizzare costruire applicativi web in breve tempo.

Symfony è inoltre equipaggiato di serie con alcuni componenti, utilizzabili anche separatamente all'infuori del framework:

- *Twig*, un motore di templating;
- *Doctrine* come ORM (*Object-Relational Mapping*);
- *Monolog* per il logging;
- *PHPUnit* per lo unit testing;
- *Swift Mailer* come componente di mailing.

Symfony fornisce inoltre supporto a altri componenti di templating e a librerie JavaScript come jQuery, TinyMCE o script.aculo.us.

Capitolo 3

Analisi

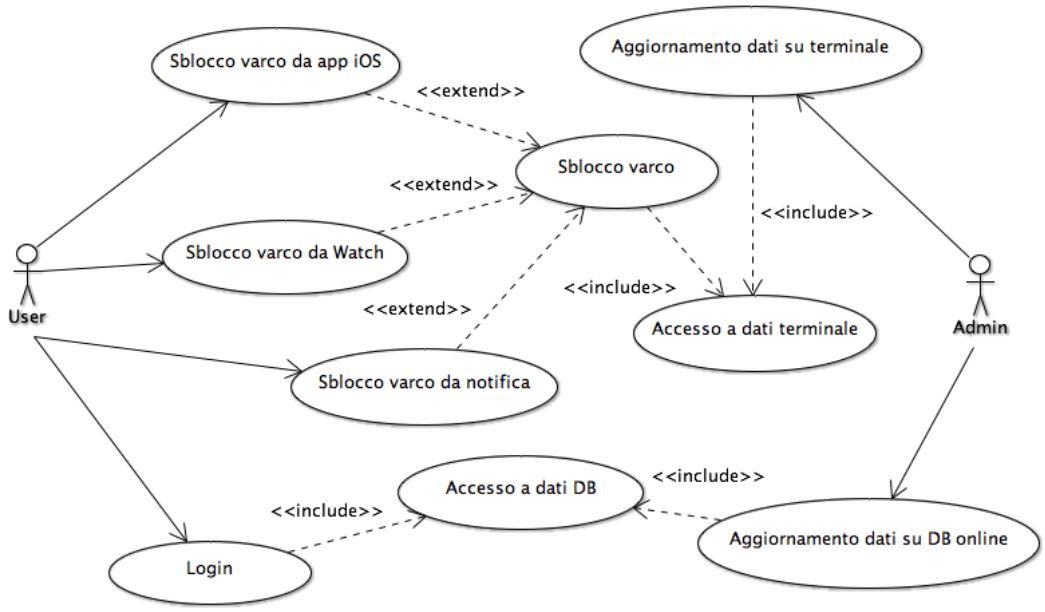


Figura 3.1: Diagramma dei casi d'uso dell'Applicazione

3.1 Casi d'uso

La figura 3.1 mostra il diagramma dei casi d'uso del progetto. Si noti che la parte relativa all'amministratore (Admin) non è stata prototipata in vista di future implementazioni da parte di Mr. APPs e Zeitgroup, perciò

tali funzioni sono attualmente gestibili dal pannello di controllo di MySQL fornito da phpMyAdmin.

Vengono quindi analizzati i singoli casi d'uso riguardanti le principali funzionalità dell'applicazione.

3.1.1 Autenticazione

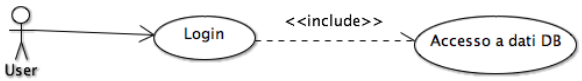


Figura 3.2: Diagramma dei casi d'uso: autenticazione

L'autenticazione dell'utente avviene tramite una chiamata a web service dove vengono passati come parametri il nome utente e l'hash SHA1 della password. Il web service a sua volta richiama il database, dove verifica la coincidenza dei dati e l'abilitazione dell'utente nel sistema, e ritorna in caso di successo una risposta contenente i dati dell'utente e i terminali al quale è abilitato.

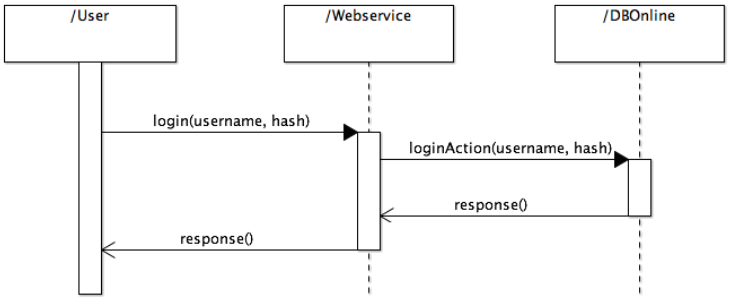


Figura 3.3: Diagramma di sequenza: autenticazione

3.1.2 Sblocco del varco

Il varco può essere sbloccato tramite applicazione iOS, tramite applicazione watchOS o tramite notifica locale ricevuta su dispositivo iOS o watchOS, come mostrato in figura 3.4.

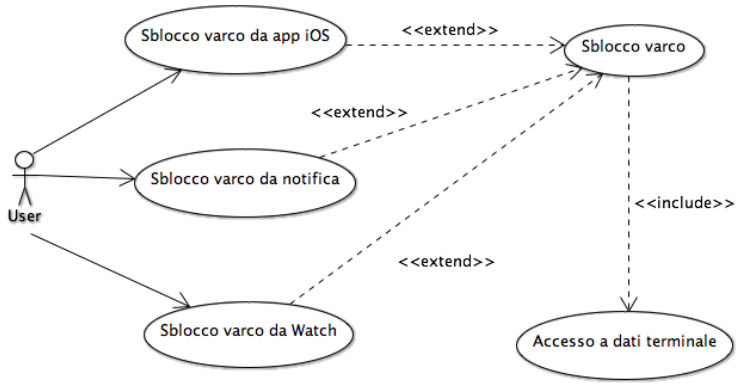


Figura 3.4: Diagramma dei casi d'uso: sblocco del varco

L'operazione di sblocco del varco può essere suddivisa in due parti: ricerca dei dispositivi nelle vicinanze e successivo collegamento al dispositivo per lo sblocco.

Ricerca dei terminali nelle vicinanze

Ad ogni apertura dell'applicazione viene avviata la ricerca dei terminali presenti nelle vicinanze, filtrando però solo quelli autorizzati dal server. Trovato un terminale autorizzato, esso viene visualizzato all'utente con un'icona verde (invece che grigia) all'interno della lista dei terminali. Muovendosi nello spazio, la lista si aggiorna in base ai terminali raggiungibili dallo smartphone.

Collegamento al terminale per lo sblocco

Il diagramma di sequenza dettagliato per questa operazione è mostrato in figura 3.5.

Selezionato il terminale desiderato dall'utente, viene inviato un messaggio di connessione alla periferica via Bluetooth, con conseguente risposta all'interno della quale vengono forniti dettagli sulla periferica. Lo smartphone quindi richiede i servizi forniti dalla periferica, che risponde con una lista di servizi.

Trovato il servizio all'interno della lista vengono richieste le caratteristiche offerte da tale servizio. Il terminale fornisce una lista di caratteristiche che non contiene tutti i dati in dettaglio: per richiedere dettagli è quindi necessario eseguire una nuova richiesta. In base al contenuto di quest'ultima

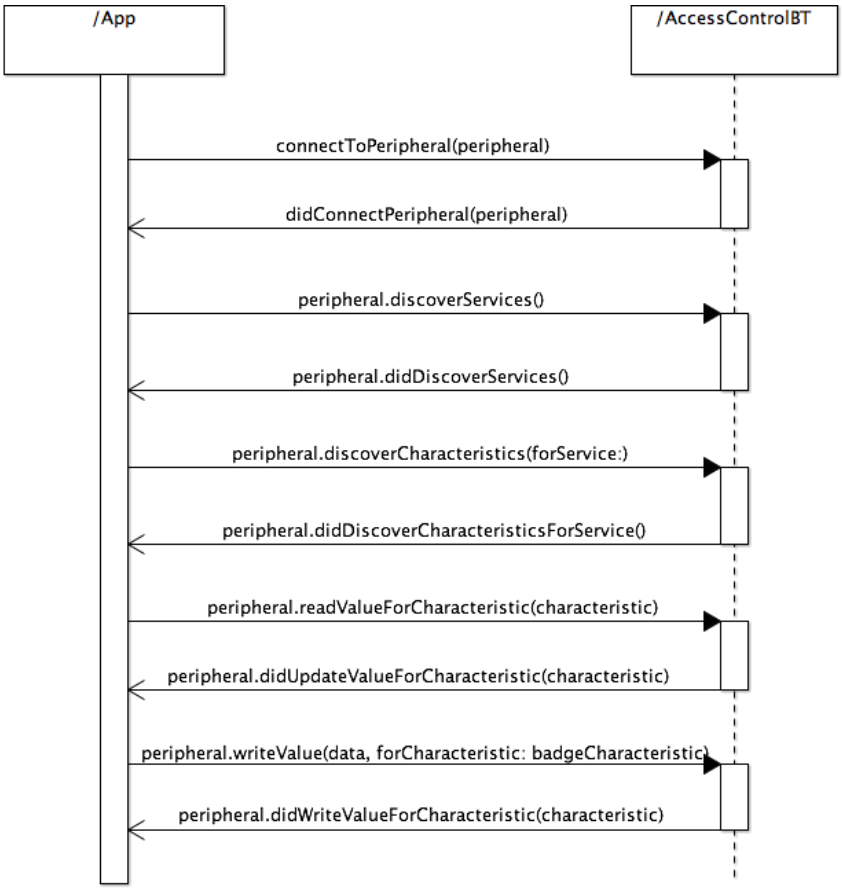


Figura 3.5: Diagramma di sequenza per lo sblocco del varco.

risposta vengono generati i dati che servono per lo sblocco della porta, quindi vengono inviati su una caratteristica diversa rispetto a quella richiesta. La risposta che ne seguirà renderà noto se il processo sia andato o meno a buon fine.

Si precisa che la risposta finale fornisce dettagli solo sull'avvenuta ricezione del segnale da parte del terminale. Non viene specificato nè l'effettivo successo o meno della chiamata, nè eventuali messaggi di errore.

Più avanti verranno analizzate in dettaglio le singole chiamate e gli algoritmi utilizzati.

Sblocco del varco da notifica

Il funzionamento dello sblocco del varco da notifica è del tutto simile a quello effettuato all'interno dell'applicazione. Le operazioni eseguite sono le stesse del collegamento al terminale; l'utente, alla ricezione della notifica, deve compiere un'azione di conferma, dopodichè viene avviato lo sblocco del varco con le stesse modalità illustrate nel paragrafo precedente.

3.1.3 Aggiornamento dati

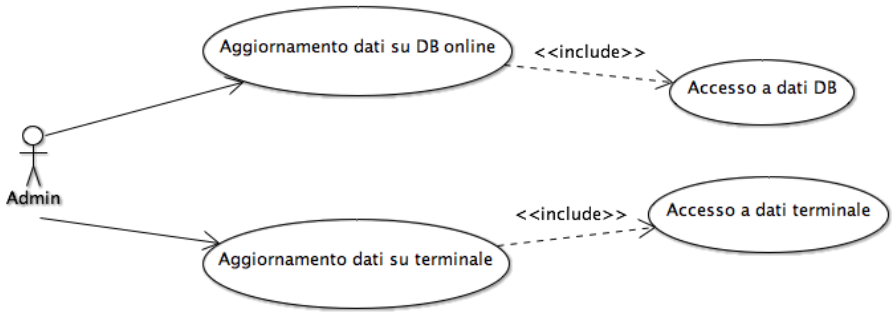


Figura 3.6: Diagramma dei casi d'uso: aggiornamento dei dati

Il diagramma mostrato in figura 3.6 fa riferimento all'aggiornamento, all'inserimento o alla cancellazione dei dati di un utente da parte del personale amministrativo. Allo stato attuale sono necessarie due operazioni di aggiornamento poichè i due database sono differenti, sia come natura che come tecnologia utilizzata. Un'implementazione futura possibile, che verrà discussa nelle valutazioni finali, sarà quella di integrare i due database o quantomeno sincronizzarli tra loro periodicamente.

3.2 Problematiche

Le principali problematiche affrontate nella fase di analisi del sistema sono state:

Permettere l'accesso alla maggior base di utenti possibile Si sta ragionando su un terminale accessi il cui compito principale è tenere

traccia dell'entrata e uscita delle persone da un varco. Finora lo strumento utilizzato per verificare l'identità è il badge o tessera magnetica, utilizzabile da qualsiasi persona. Nel caso in cui si voglia introdurre una nuova tipologia di accesso è necessario quindi garantire la retrocompatibilità in base allo strumento utilizzato e alle diverse tipologie di tecnologie utilizzate.

L'interazione dell'utente deve essere immediata L'operazione di passaggio del badge davanti al terminale è un'operazione semplice da effettuare, talvolta anche senza tirare fuori la tessera dal portafoglio (tramite NFC). Nel sistema deve essere garantita un'interazione altrettanto semplice, facilitando l'utente all'utilizzo del nuovo strumento ed impiegare il minor numero di passaggi possibile per eseguire il task. Per ragioni di sicurezza non dev'essere possibile l'operazione contraria, ovvero l'apertura del varco senza previa interazione con l'utente e autorizzazione da parte dello stesso.

Fornire un'interfaccia utilizzabile in futuro su più piattaforme Ai fini di questo progetto di tesi, il sistema è stato sviluppato per essere compatibile con iPhone e Apple Watch. È interessante tenere in considerazione, nella fase di progettazione, l'espansione futura a più piattaforme che possano supportare le tecnologie utilizzate - per fare un esempio Android, Android Wear, Windows, ecc... - in un'ottica di fruizione del sistema su più piattaforme possibili.

Sicurezza Trattandosi di uno sblocco di un varco di accesso, che in ambito aziendale può essere un portone o un tornello, è d'obbligo mantenere un alto livello di sicurezza. È necessario infatti che il sistema non permetta l'accesso a malintenzionati o persone non autorizzate.

3.3 Interfaccia utente

In base ai casi d'uso analizzati, si può facilmente constatare che le interfacce da utilizzare nell'applicazione riguarderanno l'app iPhone, l'app per Apple Watch e le notifiche.

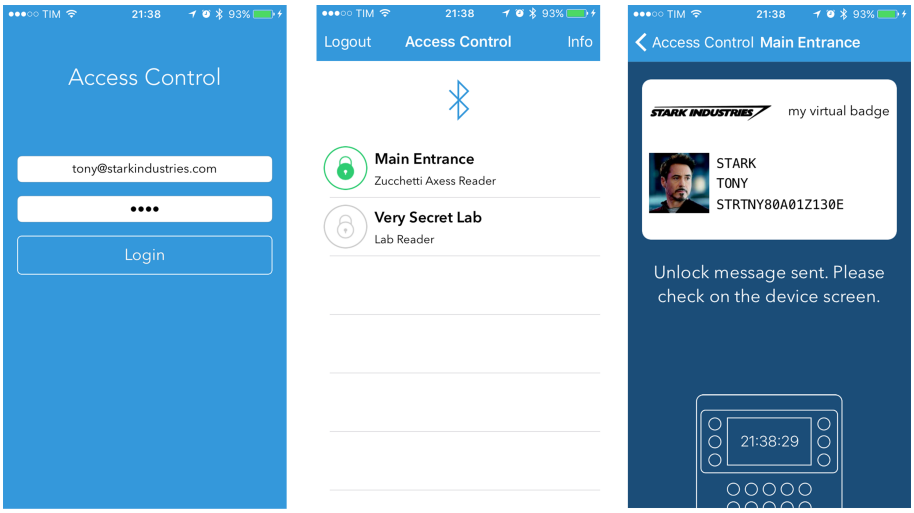


Figura 3.7: Viste dell'app iOS: autenticazione (A), lista dei varchi (B), sblocco di un varco (C)

3.3.1 iPhone

La vista (A) in figura 3.7 è dedicata all'autenticazione dell'utente. Presenta i campi dedicati all'inserimento del nome utente e della password. Facendo tap su "Login" verranno confrontati e verificati i dati con il web service. Se l'accesso è autorizzato, la schermata scomparirà per far posto alla lista dei varchi, altrimenti verrà mostrato un messaggio a video con la descrizione dell'errore riscontrato.

La lista dei varchi (B) contiene tutti i varchi autorizzati sbloccabili dall'utente. Se il varco è in zona viene evidenziato con un'icona verde, altrimenti di grigio.

Facendo tap su un qualsiasi varco si aprirà la schermata di dettaglio (C), dove viene visualizzato il badge virtuale. Qui verrà visualizzato il messaggio di avvenuta conferma dell'invio del messaggio di sblocco oppure si inviterà l'utente ad avvicinarsi al varco.

3.3.2 Apple Watch

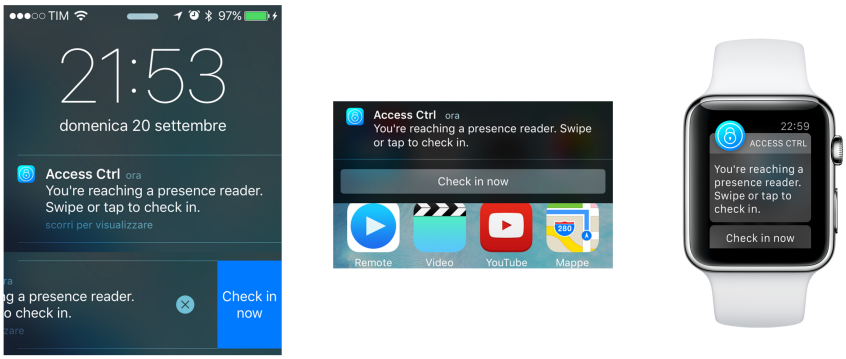
L'applicazione sarà raggiungibile tramite l'icona apposita presente nella *springboard* di watchOS, richiamabile tramite pressione della Corona Digitale

(A) (B) (C) (D)

Figura 3.8: Viste dell'app watchOS: icona dell'app (A), richiesta di autenticazione (B), lista dei varchi (C), sblocco di un varco (D)

(fig. 3.8 A). Nel caso l'utente non sia autenticato, gli sarà chiesto tramite un'apposito messaggio (B) di aprire l'app iOS e di eseguire il login. Una volta autenticato l'utente potrà vedere la lista di varchi disponibili (C), la stessa disponibile nell'app iOS. Facendo tap su un varco, verrà visualizzata la schermata (C) del badge virtuale e lo stato della richiesta.

3.3.3 Notifiche



(A) (B) (C)

Figura 3.9: Ricezione di una notifica su iPhone da schermata di blocco (A), iPhone durante l'utilizzo (B), Apple Watch (C).

Il comportamento delle notifiche è mostrato in fig. 3.9. Tutte e tre le modalità possiedono un tasto "Check in now" per eseguire lo sblocco direttamente, senza bisogno di aprire l'applicazione.

Nel caso il dispositivo iOS sia bloccato (A) la notifica verrà mostrata nella schermata di sblocco (*lock screen*) assieme alle altre notifiche; trascinando verso sinistra verrà mostrato il tasto. Nel caso il dispositivo sia in uso dall'utente (B) verrà mostrato un banner in alto che scompare da solo dopo 5 secondi. L'utente può decidere se visualizzare il banner oppure l'avviso a video tramite le impostazioni di sistema; per mostrare il tasto è necessario trascinare il banner della notifica verso il basso.

Su Apple Watch (C) la notifica è accompagnata da una vibrazione del Taptic Engine e verrà visualizzata appena l'utente rivolge lo sguardo verso il dispositivo. Il tasto è già presente nel corpo della notifica.

Capitolo 4

Progettazione e sviluppo

Lo sviluppo del progetto è stato caratterizzato da un processo incrementale di prototipazione e raffinamento. Si è mirato a sviluppare una versione funzionante di ogni singola funzionalità, per poi migliorarla e perfezionarla in un secondo tempo.

4.1 Strumenti e linguaggi utilizzati

Durante tutto il processo di sviluppo è stato fatto un utilizzo intensivo di *Git* come strumento di versionamento del codice e *Bitbucket* come servizio di hosting. Sono state utilizzate tre repository: una per l'applicazione iOS/watchOS, una per il webservice e una per la scrittura del presente documento.

Ambiente di sviluppo Sono stati impiegati Xcode 6.4 e Xcode 7.0 per la scrittura dell'applicazione iOS/watchOS e Netbeans 8.0.1 per il webservice, in esecuzione su un MacBook Pro Retina 13" con sistema operativo Mac OS X 10.10.5 *Yosemite*.

Linguaggi di programmazione L'applicazione iOS e watchOS è stata scritta in Swift 1.2 e successivamente adattata a Swift 2.0, mentre il webservice è stato scritto in PHP con l'ausilio del framework Symfony.

Database L'app iOS utilizza Realm, un DBMS e ORM open source compatibile con iOS e Android le cui fondamenta sono scritte in C++, il che lo rende molto leggero e prestante: infatti non è basato nè su

SQLite, nè su Core Data. Il webservice invece utilizza MySQL ed è amministrabile tramite MAMP PRO e phpMyAdmin.

Il web service è stato testato in locale tramite MAMP PRO utilizzando la versione di PHP 5.4.42. Il progetto Xcode si basa su CocoaPods, un gestore di dipendenze per OS X scritto in Ruby e ispirato a RubyGems. Per la creazione degli schemi UML è stato utilizzato ArgoUML.

4.2 Soluzioni adottate

L'utilizzo del sistema da parte del maggior numero di utenti è stato reso possibile grazie alla retrocompatibilità del terminale con il lettore di schede integrato e, in ogni caso, tramite le schede espandibili collegabili ad esso. Tale approccio permette l'utilizzo del sistema anche in caso di inconvenienti di vario tipo, come può essere il guasto della testina Bluetooth, di esaurimento della batteria o di smarrimento del cellulare. La retrocompabitilità può essere garantita anche a livello software, considerando il fatto che la testina Bluetooth supporta anche le versioni di Bluetooth precedenti alla 4.0.

Per la riduzione al minimo delle interazioni sono state adottate alcune accortezze. Una volta entrati nell'applicazione ed eseguita l'autenticazione (solamente la prima volta) è possibile sbloccare il terminale semplicemente facendo tap sull'elemento della lista che lo rappresenta. Lo stesso scenario si presenta nell'applicazione per Apple Watch. La situazione si semplifica ulteriormente nel caso di ricezione della notifica di prossimità, dove è possibile sbloccare il terminale direttamente facendo tap sul bottone "Check In Now" presente nel corpo della notifica. In questo caso lo sblocco è immediato, senza neanche dover aprire l'applicazione.

Il sistema è stato progettato per essere già implementabile su altre piattaforme e altri tipi di dispositivi smart. Da una parte infatti abbiamo il web service, il quale lavorando in HTTP e restituendo risposte JSON è gia in grado di servire altre piattaforme. Dall'altra l'applicazione iOS che implementa le direttive di Bluetooth 4.0 e quindi utilizza un'implementazione standard, valida anche per altre piattaforme e facile da tradurre in altri linguaggi.

Per quanto riguarda la sicurezza sono stati implementati alcuni controlli. Innanzitutto per utilizzare l'applicazione è necessario autenticarsi, perciò ogni utente autenticato è autorizzato a sbloccare solo certi varchi - in caso contrario lo sblocco verrà negato. Per sbloccare poi il terminale è necessario

essere a una distanza di circa 1 metro da esso (tale valore dipende da condizioni ambientali e dalla presenza o meno di ostacoli tra il dispositivo e il terminale). Si tenga conto che il terminale accoda le richieste multiple ed è in grado di bloccare la ricezione di nuove richieste di accesso per qualche secondo, se impostato correttamente. Inoltre i dati scambiati tra dispositivo e terminale sono cifrati tramite l'algoritmo SAFER+ (*Secure And Fast Encryption Routine*) e il segnale di sblocco è una codifica XOR del numero di badge e del valore della caratteristica ricevuta dal terminale, la quale cambia ad ogni lettura.

4.3 Dominio applicativo

4.3.1 Model

Il progetto segue il diagramma delle classi mostrato in figura 4.1.

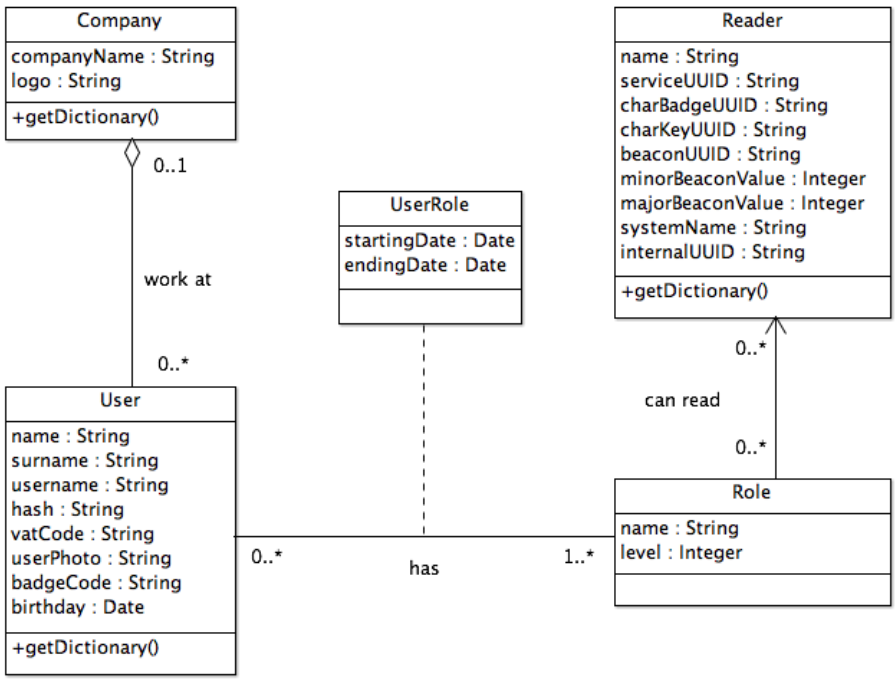


Figura 4.1: Diagramma delle classi dell'applicazione

Le classi utilizzate nell'applicazione sono:

Company

Contiene alcuni dettagli sull'azienda. In questo caso si ha:
companyName: nome dell'azienda
logo: URL al logo dell'azienda

User

Contiene tutti i dettagli dell'utente per l'identificazione e il login.
name: nome dell'utente
surname: cognome dell'utente
username: nome utente utilizzato per effettuare la login
hash: hash della password calcolato tramite SHA1
vatCode: codice fiscale
userPhoto: URL alla foto dell'utente
badgeCode: codice del badge - l'equivalente del codice inserito nel badge fisico che viene letto dal lettore
birthday: data di nascita

UserRole

Associazione che viene utilizzata per mantenere traccia della data di inizio e di fine dell'autorizzazione di un utente. Il controllo viene effettuato lato web service: se il login è effettuato correttamente si riceve la lista di ruoli al quale l'utente è abilitato.
startingDate: data di inizio autorizzazione
endingDate: data di fine autorizzazione

Role

Ruolo dell'utente in azienda, al quale verranno poi associati i lettori di rilevazione presenze.
name: nome del ruolo (p.e. "CEO")
level: livello di autorizzazione (valore numerico)

Reader

Contiene i dettagli di configurazione per l'accesso ad un lettore di rilevazione presenze.

name: nome breve assegnato al lettore
systemName: nome associato al sistema
internalUUID: identificativo assegnato dal sistema la prima volta che viene trovato *(solo app)*
serviceUUID: identificativo univoco del servizio al quale collegarsi
charBadgeUUID: identificativo univoco della caratteristica Bluetooth da utilizzare per scrivere il badge
charKeyUUID: identificativo univoco della caratteristica Bluetooth da utilizzare per criptare la chiave di sblocco
beaconUUID: identificativo univoco del beacon associato al lettore
minorBeaconValue: valore "minor" del beacon majorBeaconValue: valore "major" del beacon. Assieme a minorBeaconValue vengono utilizzati per distinguere i singoli beacon appartenenti ad un set: infatti più beacon possono condividere lo stesso UUID ma essere caratterizzati da diversi valori di major e minor.

Tale modello vale per il database online utilizzato dal webservice. Lato applicazione invece viene semplificato, poichè l'app accetta solamente un utente autenticato per volta, quindi le associazioni tra *Company* e *User* diventano di tipo 1-1 (l'utente al momento del login è associato ad una sola azienda). L'associazione tra *User* e *Role* diventa di tipo 1-n, poichè esiste solo un utente autenticato al quale vengono attribuiti n ruoli.

Sebbene la struttura sia snella e molto semplice, permette di gestire già i ruoli, ovvero: un'utente possiede uno o più ruoli, i quali permettono di sbloccare varchi diversi. Per fare un esempio, un addetto di laboratorio può sbloccare varchi che un normale operaio non potrebbe perchè di diverso livello e con diverse mansioni.

4.3.2 View

Le viste delle app iOS e watchOS sono state create tramite Interface Builder, lo strumento di sviluppo di interfacce presente all'interno di Xcode.

View di iOS

L'applicazione iOS presenta tre viste principali:

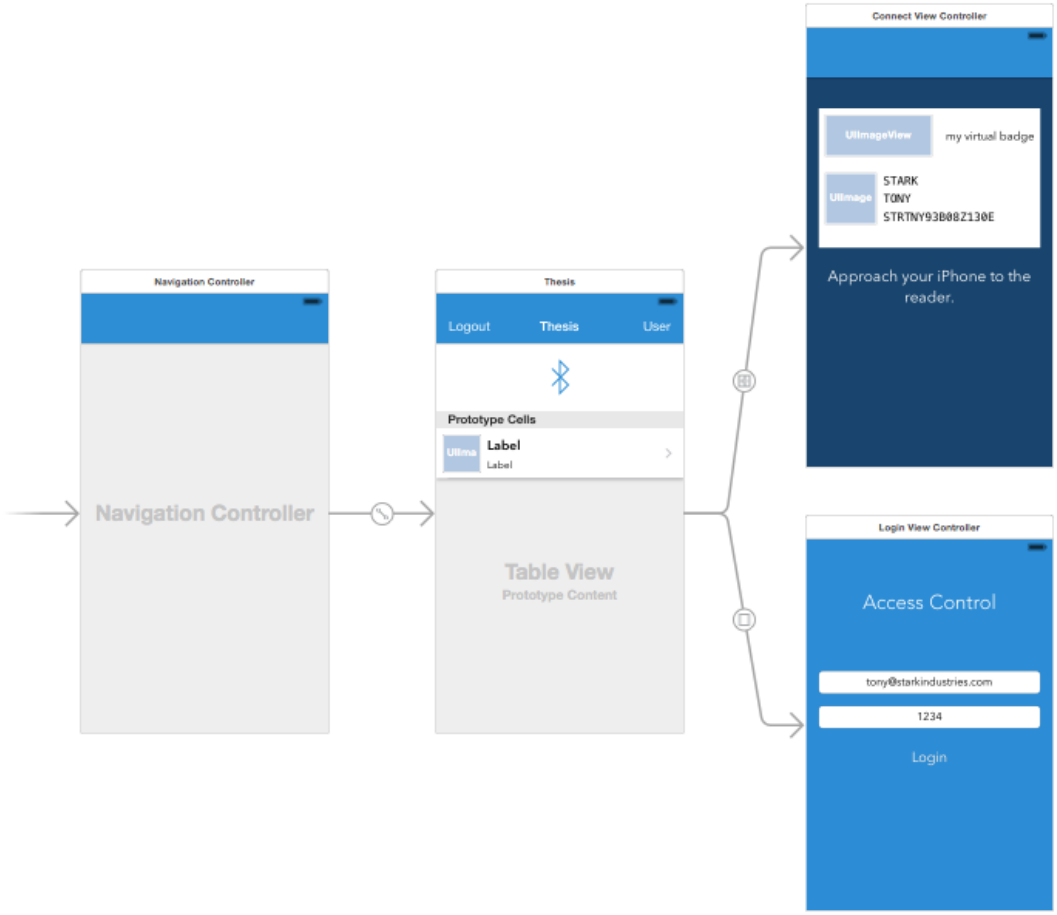


Figura 4.2: Storyboard visivo dell'app iOS

ViewController, la vista principale con la lista di tutti i lettori autorizzati allo sblocco. All'interno della tabella principale le celle sono di tipo HomeCell, derivate da `UITableViewCell`.

LoginViewController, vista di login. Viene chiamata se l'utente non ha effettuato l'autenticazione o se l'utente esegue il logout dall'applicazione.

ConnectViewController, vista che si occupa di fornire all'utente informazioni sul badge personale dell'utente e sullo stato di avanzamento della richiesta di sblocco.

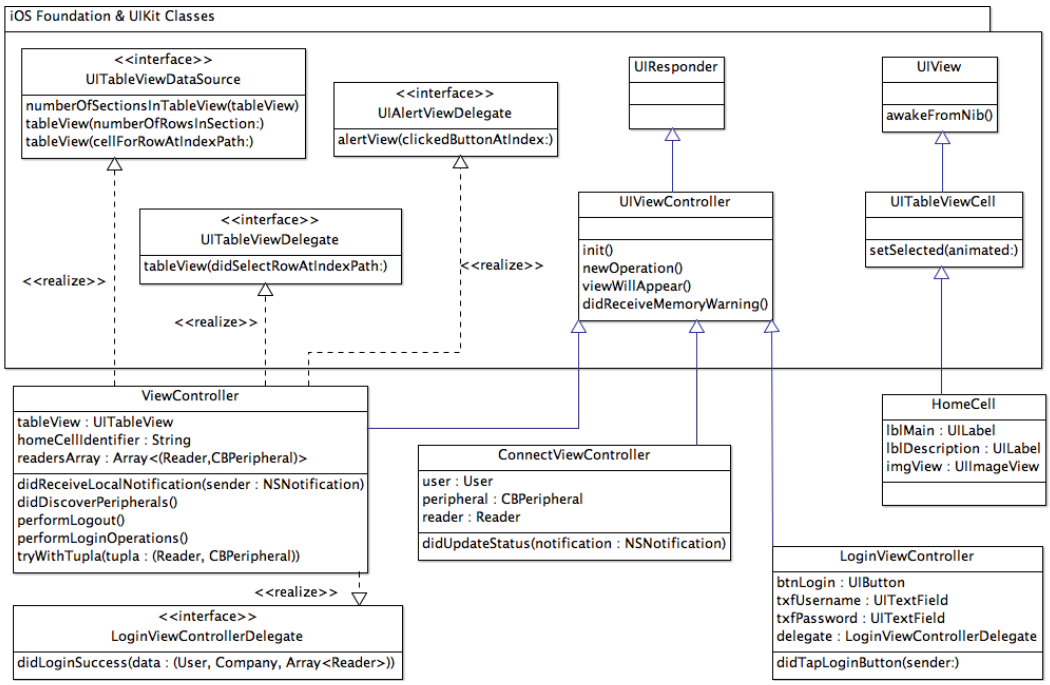


Figura 4.3: Diagramma delle classi: viste iOS

View di watchOS

Le viste di watchOS differiscono leggermente da quelle di iOS. Innanzitutto non è possibile eseguire l'autenticazione, per cui se un utente non è autenticato viene visualizzato un messaggio in cui viene invitato ad aprire l'applicazione su iPhone per effettuare l'azione; di conseguenza non è possibile eseguire il logout.

Una volta caricata la lista di lettori disponibili (vista InterfaceController), per tentare l'apertura di un varco è sufficiente fare tap sul lettore desiderato. Si aprirà quindi una schermata riepilogativa (WatchConnectController) simile al ConnectViewController presente nella controparte iOS.

4.3.3 Controller

iOS e watchOS

Sull'app sono stati adottati alcuni controller, i quali svolgono operazioni mirate al singolo task - per migliorare l'ordine e la leggibilità del codice - e

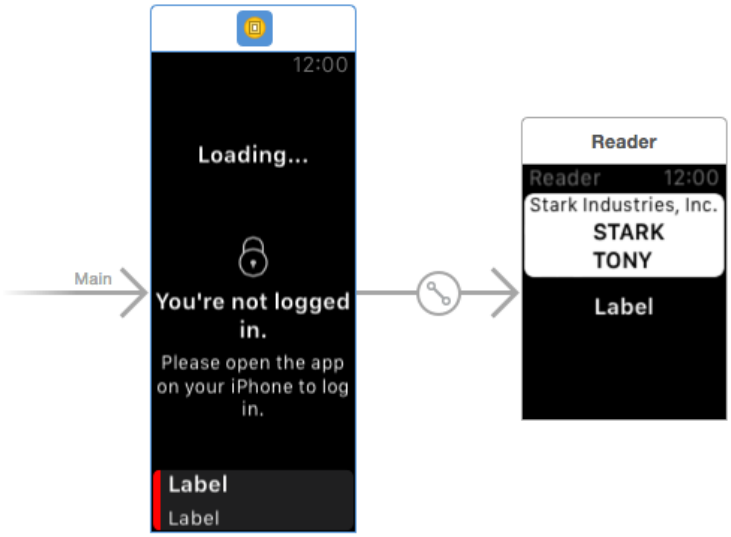


Figura 4.4: Storyboard visivo dell'app watchOS

sono stati sviluppati adoperando diversi pattern. Il diagramma in figura 4.5 mostra gli attributi e i metodi di ciascuna classe.

BluetoothController Gestisce la logica della connessione via Bluetooth, includendo quindi la ricerca di dispositivi, il collegamento, la ricerca di profili GATT e la lettura/scrittura di caratteristiche. Tale controller si interfaccia con la testina Bluetooth collegata al terminale Zucchetti. Le API utilizzate in questo controller sono quelle dell'SDK Core Bluetooth di Apple. Implementa il pattern Singleton.

BeaconController Gestisce la scansione dei beacon presenti in zona associati ai lettori abilitati per l'utente. In caso di ritrovamento di un beacon corretto, il controller emette una notifica locale (`UILocalNotification`) all'avvicinamento del dispositivo al beacon. Utilizza l'SDK Core Location di Apple. Implementa il pattern Singleton.

WebController Si occupa dell'esecuzione della chiamata di login. Nonostante abbia una sola funzione è possibile aggiungere nuove chiamate molto semplicemente utilizzando il metodo `operationWithPath:`. Si basa su AFNetworking, una potente libreria per il networking disponibile per iOS e Mac OS X. Implementa il pattern Singleton.

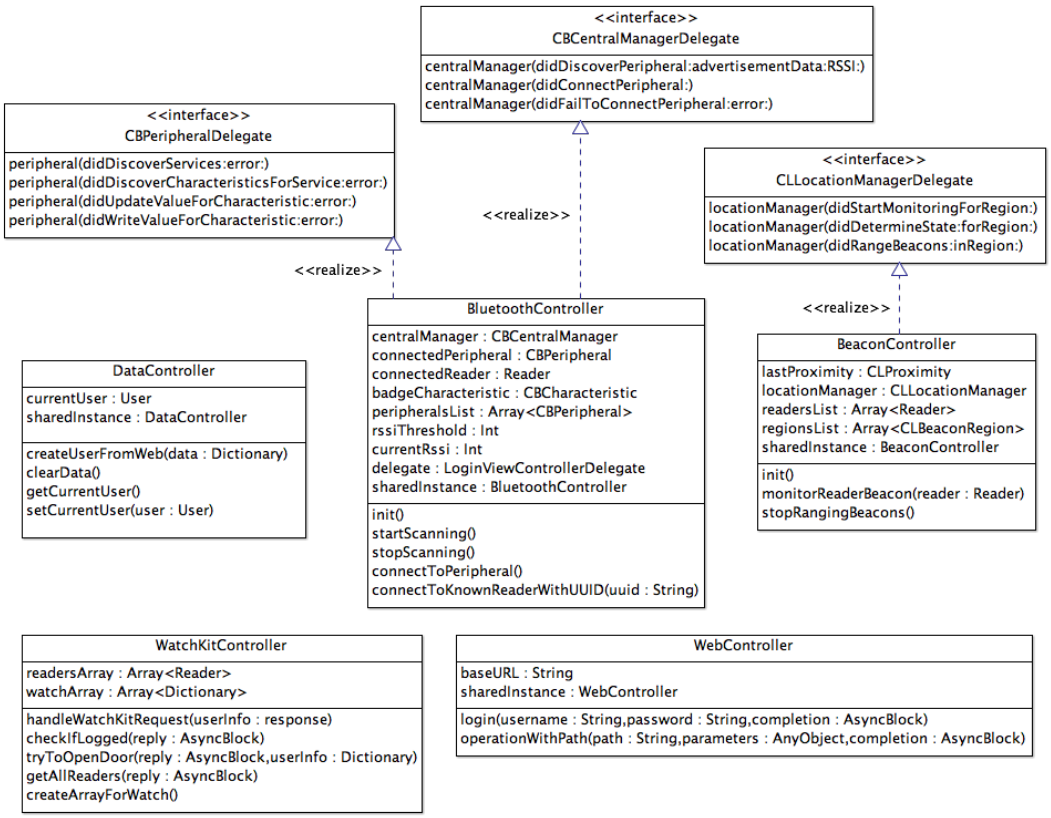


Figura 4.5: Diagramma delle classi: controller

DataController Si occupa dell'aggiunta dei dati provenienti dal web service al database interno, utilizzando Realm. Mantiene il riferimento all'utente corrente e cancella i dati in caso di logout. Implementa il pattern Singleton.

WatchKitController Riceve le chiamate dal metodo `application: handleWatchKitExtensionRequest: reply:` di AppDelegate, esegue le istruzioni richieste e le restituisce tramite un blocco asincrono.

È presente inoltre la classe statica Utils che contiene alcune funzioni utilizzate all'interno dell'app - esecuzione ritardata dei metodi, criptaggio badge e generazione dell'hash SHA1 per la password.

Web service

Il web service, considerato il fatto che esegue solamente l'autenticazione, presenta un'unico controller (DefaultController). Esso presenta quindi un solo metodo, richiamabile tramite la rotta `/api/login`.

I parametri di tale chiamata sono il nome utente e l'hash SHA1 della password. Una volta controllata la correttezza dei dati e la valitidità dell'utente, vengono recuperati i dati dal database e restituiti in formato JSON.

All'interno di un progetto Symfony, il modello - chiamato "Entity" - possiede anche delle classi intermedie dette "Repository" che si occupano dell'esecuzione di query relative ad una determinata Entity. In questo caso è stata utilizzata la classe UserRepository per permettere di ritrovare le informazioni di un utente partendo dai suoi dati di autenticazione.

4.4 Architettura del sistema

La figura 4.6 mostra l'architettura generale del sistema. Al centro del sistema troviamo l'app iOS (iPhone), la quale si collega tramite Bluetooth 4.0 al terminale di controllo degli accessi per lo sblocco del varco, al beacon per la localizzazione e l'invio delle notifiche nel caso l'app sia chiusa o in esecuzione in background e ad Apple Watch per l'esecuzione della logica dell'applicazione watchOS. Come già discusso, l'app iOS si collega al web service tramite protocollo HTTP per l'autenticazione e il download dei dati necessari per il funzionamento del sistema.

4.5 Comunicazione

4.5.1 iPhone - Terminale Accessi (testina Bluetooth)

La comunicazione tra iPhone e il terminale accessi avviene utilizzando le API del framework nativo Core Bluetooth fornito da Apple. Tutta la logica di questa comunicazione è contenuta all'interno della classe BluetoothController del progetto Xcode. Tale classe contiene un oggetto di tipo `CBCentralManager` (che chiameremo "gestore") il quale si occupa della comunicazione a basso livello. Chiameremo "delegati" (*delegate*) i metodi dell'interfaccia di `CBCentralManager` che sono esposti pubblicamente.

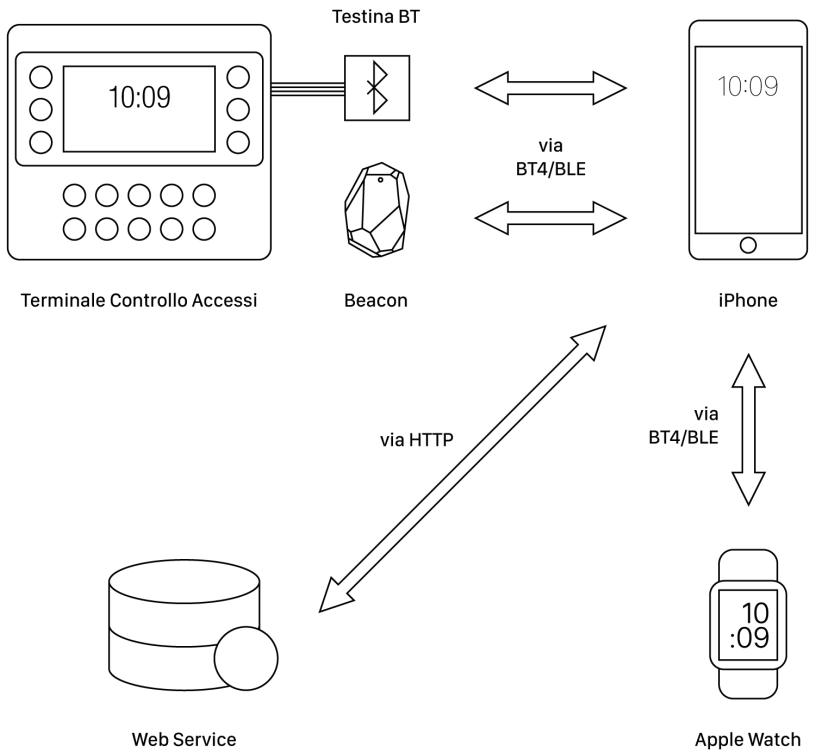


Figura 4.6: Architettura del sistema

Una volta avviata la ricerca dei dispositivi associati all'utente disponibili in zona tramite il metodo `scanForPeripheralsWithService: options:` il gestore chiama il metodo delegato `centralManager: didDiscoverPeripheral: advertisementData: RSSI`, il quale filtra i dispositivi disponibili in base alla lista ritornata dal server.

La prima volta che viene rilevato un dispositivo Bluetooth, il sistema gli assegna un UUID. Ciò è molto utile poichè le connessioni successive al dispositivo possono avvenire tramite la chiamata diretta `retrievePeripheralsWithIdentifiers:` di `CBCentralManager`, senza dover eseguire ogni volta la scansione dei dispositivi in zona, rendendo la procedura più veloce e stabile. Questo UUID viene salvato localmente nel campo internalUUID della classe Reader.

Il funzionamento dello sblocco del varco è mostrato in figura 3.5. L'u-

tente sceglie il dispositivo (*peripheral*) a cui collegarsi richiamando il metodo `connectToPeripheral`. Il delegato risponde con i dettagli del dispositivo, dopodichè si da inizio alla ricerca dei servizi (*services*) forniti da esso.

Alla ricezione dei servizi si ricerca quello identificato dall'UUID del campo serviceUUID dell'oggetto Reader attualmente in esame. Si ricercano quindi le caratteristiche di tale servizio. Trovate le caratteristiche, viene richiesta la lettura completa della caratteristica identificata dall'UUID specificato nel campo charKeyUUID dell'oggetto Reader.

Ricevuti i dati sulla caratteristica, si procede alla scrittura. All'interno della caratteristica specificata dall'UUID del campo charBadgeUUID di Reader, viene scritto il valore attuale della caratteristica appena trovata (che cambia ad ogni lettura) al quale viene effettuata un'operazione bitwise XOR con l'identificativo del badge associato all'utente (campo badgeCode di User). Il listato di tale funzione è allegato qui di seguito.

Se la scrittura va a buon fine (metodo `peripheral: didWriteValueFor Characteristic: error`) viene restituito un messaggio di successo, altrimenti viene restituito un errore. Dopodichè vengono deallocate le variabili globali utilizzate e la periferica viene disconnessa.

Listing 4.1: Criptaggio del badge

```
1  internal static func libMessageCripter(source: [UInt8], key: [UInt8
       ]) -> [UInt8?]{
2          let srcLength : Int = source.count
3          var result : [UInt8?] = [UInt8?](count: srcLength,
               repeatedValue:nil)
4          var cnt : Int = 0
5          for var counter : Int = 0; counter < srcLength; ++counter {
6                  result[counter] = (source[counter] ^ key[cnt])
7                  ++cnt
8                  cnt = (cnt < key.count) ? cnt : 0
9          }
10         return result
11 }
```

4.5.2 iPhone - Beacon

La comunicazione tra iPhone e Beacon avviene tramite le API fornite dal framework Core Location. Tale framework è dedicato a tutto ciò che riguarda la posizione dell'utente nello spazio, quindi non solo geograficamente ma anche localmente [13]. Nonostante anche i beacon utilizzino Bluetooth 4.0, Apple ha deciso di includere tali API in Core Location anzichè in Core Bluetooth per poter fornire allo sviluppatore uno strumento in più per conoscere la posizione dell'utente utilizzando concetti già visti in precedenza con il contesto delle mappe.

La ricerca e la connessione ad un beacon avvengono in due fasi. In un primo stadio si ricerca il beacon all'interno dell'area monitorabile dal dispositivo, chiamata *region*. Dopodichè è possibile ricevere i dettagli esatti sul beacon desiderato (*ranging*) come distanza dal beacon, nome, accuratezza e RSSI. Tale accorgimento è stato probabilmente introdotto per risparmiare risorse energetiche: ricordiamo che queste operazioni richiedono un controllo costante dei beacon presenti in zona che può consumare molto facilmente la batteria dello smartphone.

Il controller utilizzato per la gestione dei beacon è BeaconController. All'interno troviamo i metodi delegati di `CLLocationManager`, il gestore fornito da Core Location.

Una volta trovato un lettore abilitato viene immediatamente avviato il monitoraggio del beacon associato tramite il metodo `monitorReaderBeacon:`. Nel caso l'applicazione non abbia i dovuti permessi vengono richiesti a video dal sistema operativo. Trovato il beacon nella regione viene chiesto il suo stato, ovvero se è all'interno o all'esterno della regione. Se è all'interno viene avviato il ranging del beacon, altrimenti viene terminato.

Avviato il ranging, tramite il metodo `locationManager: didRangeBeacons: inRegion:` vengono aggiornati i dettagli sul beacon. Qui poi viene preso in considerazione solo il beacon desiderato e, se esso è in zona, viene inviata una notifica locale (`UILocalNotification`).

Si consideri che la notifica viene emessa solo se si passa da una prossimità di tipo "Far" (distante) ad una prossimità di tipo "Near" (nelle vicinanze) o "Immediate" (vicino). L'aspetto delle notifiche è mostrato nell'immagine 3.9.

Core Location consente di ricevere la notifica locale anche se l'applicazione è chiusa oppure in esecuzione in background. In questi casi, però, la

rilevazione del beacon non è immediata e varia in base allo stato attuale dell'applicazione: per esempio, se l'app è chiusa, possono volerci dai 30 ai 45 secondi circa per l'invio della notifica.

4.5.3 iPhone - Apple Watch

Apple Watch fa uso di WatchKit per la comunicazione tra iPhone e Apple Watch. WatchKit è un framework nativo presente su entrambi i dispositivi e gestisce la logica, le schermate e la comunicazione.

All'interno del progetto Xcode l'applicazione per Apple Watch è considerata un'*estensione*, un concetto introdotto da iOS 8. L'estensione è una parte di progetto a sé stante con relativo binario generato che può comunicare con l'applicazione iOS e con altre estensioni. Assieme ad iOS 8 sono stati presentati alcuni tipi di estensione il cui obiettivo è far cooperare tra loro le applicazioni in un ambiente sicuro. Tra queste troviamo le tastiere personalizzate, le *Action Extension* per permettere all'utente un'azione con la propria applicazione da app terze e le *Today Extension* per aggiungere il widget dell'applicazione al Centro Notifiche.

La comunicazione tra i due dispositivi avviene tramite Bluetooth 4.0; non è possibile sapere in dettaglio il funzionamento a basso livello della comunicazione (per esempio handshake, messaggi, polling, etc...) poichè non sono disponibili informazioni a riguardo da parte di Apple. Allo sviluppatore vengono messi a disposizione due metodi, uno per watchOS dove vengono inviati i dati e uno per iOS dove vengono ricevuti.

Infatti, ogni qualvolta si voglia richiamare l'applicazione iOS da watchOS è necessario utilizzare il metodo della classe `WKInterfaceController` `openParentApplication: reply:`, il quale accetta come primo parametro un oggetto qualsiasi e come secondo parametro un blocco asincrono. Si può quindi passare come primo parametro un dizionario in cui viene specificata l'operazione da eseguire. Per esempio, volendo controllare se l'utente è loggato, chiameremo dall'applicazione watchOS:

Listing 4.2: Richiesta di dati dall'app Apple Watch

```
1 InterfaceController.openParentApplication(["request":"isLogged"] as
      [NSObject:AnyObject], reply: { (reply, error) -> Void in
2       /* questo è il codice del blocco che viene richiamato una
           volta che ho completato le operazioni da eseguire su iOS.
            */
```

```
3 })
```

Come si può notare il blocco asincrono ritorna due parametri: *reply* e *error*. Il primo è valorizzato nel caso la chiamata abbia successo, il secondo nel caso ci siano degli errori.

> I blocchi asincroni, introdotti in C e di conseguenza in C++, Objective C e in ultima battuta su Swift, permettono di dichiarare metodi di callback in linea che vengono poi richiamati al completamento di una determinata operazione asincrona in esecuzione su uno o più thread. È possibile implementare il meccanismo dei blocchi asincroni anche tramite *Grand Central Dispatch* (GCD), la tecnologia open source di Apple per la gestione del multithreading. Nell'appendice A verrà illustrata una breve introduzione alla gestione della concorrenza tramite GCD.

Nella controparte iOS, il metodo `openParentApplication: reply:` richiama uno dei metodi di AppDelegate, il punto di entrata dell'applicazione. Tale metodo si chiama `application: handleWatchKitExtensionRequest: reply`, dove ritroviamo i dati inviati da watchOS (primo parametro) e il blocco asincrono (secondo parametro). All'interno di questo metodo è necessario controllare la richiesta ricevuta, effettuare le opportune operazioni e richiamare il blocco asincrono con i dati richiesti.

Per semplificare il codice e renderlo più chiaro si è deciso di creare un controller dedicato all'interno dell'applicazione iOS, WatchKitController, liberando così l'AppDelegate da codice inappropriato per il compito svolto. Il metodo diventa così una chiamata a WatchKitController:

Listing 4.3: Metodo delegato di WatchKit nell'AppDelegate dell'applicazione

```
1 func application(application: UIApplication,
    handleWatchKitExtensionRequest userInfo: [NSObject : AnyObject
    ]?, reply: (([NSObject : AnyObject]!) -> Void)!) {
2     watchController.handleWatchKitRequest(userInfo, response:
          reply)
3 }
```

All'interno di WatchKitController il metodo `handleWatchKitRequest: response:` gestisce le varie chiamate e chiama i metodi relativi. Ogni metodo richiede come parametro il blocco asincrono, così da chiamarlo diret-

tamente una volta eseguite tutte le operazioni. Le chiamate attualmente implementate sono:

isLogged verifica se l'utente è autenticato al sistema. Se non è autenticato, nell'applicazione watchOS viene visualizzato un messaggio che invita l'utente a eseguire l'autenticazione sull'app iOS;

allReaders invia la lista dei lettori disponibili all'app watchOS;

openDoor invia il segnale di sblocco del varco selezionato all'app iOS.

L'esecuzione di questi metodi può avvenire anche ad applicazione chiusa o in esecuzione in background. Per ogni richiesta effettuata dall'app watchOS, la controparte iOS si apre in background per qualche istante ed esegue solamente il metodo `application: handleWatchKitExtensionRequest: reply`. Naturalmente all'utente finale questo aspetto è totalmente invisibile.

In questa sezione sono state analizzate richieste effettuate esclusivamente dall'app watchOS all'app iOS. Per permettere il contrario, ovvero scambiare messaggi da iOS a watchOS, si può scegliere tra diversi approcci:

NSUserDefaults con suite: `NSUserDefaults` è una classe molto utilizzata su iOS per il salvataggio in memoria di dati di piccole dimensioni, tipicamente valori e parametri utili all'interno dell'applicazione. Inizializzando un oggetto `NSUserDefaults` con il parametro `suiteName` è possibile condividere tali dati con le estensioni. Aggiungendo degli osservatori (*observers*) sull'app watchOS sarà possibile controllare quando tali dati cambiano di valore.

Darwin Notifications: è l'implementazione a basso livello di `NSNotificationCenter`, una classe di iOS utilizzata per notificare cambiamenti all'interno dell'applicazione (paragonabile all'utilizzo di un pattern *observer*). Come intuibile dal nome, utilizza metodi a basso livello forniti da Darwin, le fondamenta su cui sono costruiti OS X e iOS. È incluso all'interno del framework Core Foundation.

MMWormhole: realizzato da Mutual Mobile Inc., è un wrapper per le notifiche Darwin. Utilizza un approccio molto simile a quello appena visto nell'app, però bidirezionale: è possibile inviare dati da entrambe le parti.

Per l'implementazione di questi approcci è prima necessario registrare un App Group, ovvero un identificativo dell'intera applicazione che includa anche le estensioni e le app WatchKit. Tale identificativo è registrabile sul sito Apple Developer nella sezione Certificati ed è da specificare all'inizializzazione dei componenti appena visti.

4.5.4 iPhone - Web Service

Per eseguire l'autenticazione all'interno dell'applicazione è necessario eseguire una chiamata HTTP al web service. Tale web service è in esecuzione in una rete locale (*LAN*) sulla porta 80 del server MAMP. È possibile comunque spostare il software su un server online che supporti Symfony.

Lato server

Ogni chiamata di Symfony è impostata su una rotta (*route*). L'unica rotta creata per questo progetto è `/api/login`: ciò significa che si sta richiedendo il metodo "login" appartenente al set di chiamate "api" sul server in esecuzione. Tale richiesta necessita di due parametri in entrata, passabili tramite metodo GET: username e password (quest'ultimo in realtà è l'hash SHA1 della password inserita dall'utente).

Supponendo che il server sia in esecuzione sull'indirizzo IP locale 192.168.0.245, l'URL finale da contattare per eseguire l'autenticazione con nome utente `tony.stark` e password `1234` sarà `http://192.168.0.245/api/login?username=tony.stark&password=7110eda4d09e062aa5e4a390b0a572ac0d2c0220`.

La rotta viene intercettata dal metodo `loginAction` di DefaultController. Questo metodo si occupa di ricavare i parametri dalla richiesta e ricercare, tramite il metodo `getUserDetails` di UserRepository, la presenza dell'utente nel database e la sua validità. Si ricorda infatti che per essere autenticato l'utente deve essere associato ad un ruolo da una data d'inizio minore e una data di fine maggiore rispetto a quella odierna. La query risultante, infatti, sarà: `SELECT role FROM AppBundle:Role role JOIN role.usersRoles ur WHERE ur.user = :user AND ur.startingDate <= :date AND ur.endingDate >= :date`. Tale linguaggio è chiamato Doctrine Query Language (*DQL*) e viene utilizzato da Doctrine esclusivamente per la composizione di query testuali (non effettuate tramite ORM).

Viene poi composta la risposta da restituire all'applicazione. Tale risposta si basa su uno scheletro così composto:

Listing 4.4: Esempio di risposta JSON dal server

```
{
        "success":true/false,
        "data":{...},
        "message":"..."
}
```

Se l'utente non è autorizzato viene restituito un messaggio d'errore (campo *message*), altrimenti l'esecuzione del metodo continua. In caso di errore è valorizzato il campo *message*, mentre *data* è impostato a `null`; viceversa in caso di successo.

In caso di successo, vengono restituiti i dettagli sull'utente, sull'azienda e i ruoli associati. All'interno dei ruoli sono presenti i dettagli sui terminali autorizzati. Tutti gli oggetti vengono poi convertiti in array associativi (dizionari) chiave-valore.

Il formato di output è JSON: a tal proposito viene utilizzato l'oggetto JsonResponse di Symfony che si occupa automaticamente di serializzare la risposta. È necessario inoltre impostare il campo `Content-Type` dell'intestazione HTTP sul tipo MIME `application/json`.

Una richiesta che va a buon fine presenta i seguenti dati nel campo *data*:

Listing 4.5: Esempio di risposta JSON dal server. Dettaglio sui dati ricevuti.

```
{
        "user": {
                "id":1,
                "name":"Tony",
                "surname":"Stark",
                ...
        }, "company": {
                "id":1,
                "companyName":"Stark Industries Corp.",
                ...
        }, "roles": [
                {
                        "id":1,
                        "level":10,
                        "name":"Chief Executive Officier",
                        "readers":[ {
                                        "id":1,
```

```
18                                      "name":"Main Entrance",
19                                      ...
20                              },{
21                                      "id":2,
22                                      "name":"Very Secret Lab",
23                                      ...
24                              }, ...
25                      ]
26              }, ...
27      ]
28 }
```

Lato client

Il client esegue l'autenticazione ogni volta che l'applicazione viene aperta. Questo per ragioni di sicurezza: prendendo come esempio lo scenario di un dipendente licenziato o non più autorizzato ad accedere a certe aree, è necessario bloccare il prima possibile la possibilità di accedere dal client.

Tale autenticazione avviene tramite il metodo `login: username: password: completion:` della classe WebController. Questo metodo non fa altro che preparare i parametri da inviare (username e SHA1 della password) e richiamare il metodo `operationWithPath: parameters: completion: error:` che si occupa di eseguire la richiesta e di ricevere la risposta.

Alla ricezione di una risposta con successo, vengono salvati i dati sul database e viene chiamato il blocco asincrono `completion:`. In caso di errore il campo *data* non è valorizzato, mentre il campo *error* ritorna la descrizione dell'errore inviata dal server.

Capitolo 5

Collaudo e valutazione complessiva del sistema

Lo sviluppo del sistema si è basato sui requisiti descritti in questo documento. L'implementazione di tali funzionalità è avvenuto con successo e il prototipo realizzato consente di già di svolgere le operazioni basilari attualmente offerte dal badge fisico. Per l'utilizzo di alcune funzioni (es. scelta della causale, inserimento PIN) è ancora necessario interagire con il terminale. Lo sviluppo è durato circa tre mesi e si è svolto per la maggior parte in azienda.

Il prototipo è quindi pronto per essere testato all'interno di una realtà aziendale abbastanza piccola, caratterizzata da un unico terminale. Prerogativa per una prima distribuzione di test è la creazione di un'app simile anche per Android, così da usufruire già da subito di una fetta di utenza più ampia rispetto ad iOS.

5.1 Test effettuati

I test sono stati effettuati sul terminale di controllo accessi Zucchetti fornito da Zeitgroup. Avendo a disposizione solo un esemplare del dispositivo non è stato possibile provare il software interagendo su più terminali contemporaneamente; sono state comunque adottate scelte di progettazione e accortezze per poter gestire più di un terminale e, allo stesso tempo, per evitare l'accesso contemporaneo al varco da parte di più persone.

È stata collaudata intensivamente l'operazione di sblocco tramite tutte e tre modalità - da iPhone, da Apple Watch e da notifica su iOS e watchOS. Il sistema è stato testato solo su rete locale poichè il server era in esecuzione sul computer tramite MAMP.

5.2 Miglioramenti e sviluppi futuri

Il sistema può essere potenziato e migliorato, essendo quello presentato un primo prototipo. Le due aziende, Mr. APPs e Zeitgroup, sono già al lavoro sull'ideazione e sulla progettazione di un prodotto finale partendo dal presente prototipo. Questi alcuni tra i punti consigliati:

Integrazione iBeacon nella testina Bluetooth

La testina Bluetooth 4.0 è dotata di un microprocessore ARM Cortex M0 e di un chip Bluetooth. Gli stessi tipi di componenti sono utilizzati dal beacon Estimote. Come già discusso, le due schede si differenziano solo per il fatto che il beacon supporta i suoi protocolli dedicati (iBeacon e Eddystone), mentre la testina fornita no. Un passo in avanti in tal senso sarebbe quello di integrare i due componenti in uno solo. Si ricorda che, mentre Eddystone è un protocollo open source (ma più recente e quindi meno supportato), iBeacon prevede l'adesione del produttore hardware al programma di licenza "MFi" di Apple.

Integrazione tra il database del terminale e il database online

Attualmente i database del terminale e del web service sono separati, sia per il livello di prototipazione che per l'utilizzo di due ambienti diversi. Il database del terminale, infatti, si basa sul file system e gestisce record memorizzati all'interno di file testuali, mentre il web service si basa su PHP e Symfony e su un database MySQL generato da Doctrine.

Una soluzione tampone può essere quella di creare un *cronjob* (operazione pianificata) che sincronizzi i due sistemi periodicamente, ovvero che legga i dati dal database del webservice, li converta in un file .txt conforme alle direttive specificate nel documento [8] e li carichi sul server interno del terminale tramite FTP. Il terminale infatti è aggiornabile sia tramite pannello web via HTTP che tramite collegamento via FTP.

Il passo successivo sarà creare un pannello di controllo web dove il titolare o gli addetti amministrativi possano impostare facilmente i parametri di funzionamento e i permessi di accesso ai singoli terminali.

Gestione degli orari e delle fasce orarie di accesso

Un aspetto importante è la gestione degli orari di ingresso e uscita, che attualmente avviene internamente sul terminale. Un'implementazione può essere la visione degli orari di entrata e uscita dei dipendenti e la gestione di fasce d'orario d'accesso direttamente in tempo reale da remoto tramite pannello web. Inoltre, la scelta di timbrare l'entrata o l'uscita avviene sul terminale premendo l'apposito tasto. Si potrebbe eliminare questo passaggio rendendolo intelligente, controllando gli orari e fissando delle fasce d'orario in cui avviene l'accesso e l'uscita, oltre all'impostazione di causali speciali (es. ritardo giustificato, pranzo, permesso, riunione, ecc...).

Indipendenza dal terminale

In casi particolari il terminale può essere rimosso per permettere una gestione interamente lato client. I casi possono essere due: l'utilizzo del software come solo sistema di timbratura oppure il collegamento di un relè Bluetooth (fig. 5.1) per lo sblocco della porta.

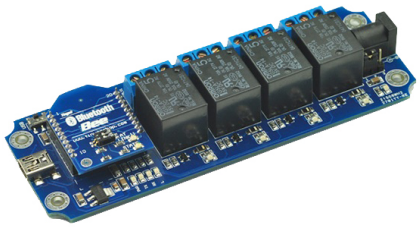


Figura 5.1: Un esempio di relè Bluetooth.

Nel primo caso può essere utilizzato il beacon come strumento per conoscere la posizione dell'utente e autorizzarlo all'accesso, oltre per inviargli notifiche all'entrata e all'uscita dall'edificio. Nel secondo caso il terminale può essere sostituito da un più semplice relè dotato di interfaccia Bluetooth pilotabile da qualsiasi dispositivo conforme. In entrambi i casi la registrazio-

ne della presenza avviene dal client e viene gestita comunicando al server i dati appena ricevuti.

Applicazione aziendale

Le funzionalità implementate e descritte finora possono fare parte di un contesto più ampio che possa comprendere altre funzionalità utili per il personale aziendale. Tra queste potremmo trovare:

- notizie e comunicazioni, con relative notifiche sia programmate che basate sulla localizzazione (inviando quindi una notifica all'entrata o all'uscita dal varco);
- un calendario aziendale con riunioni ed eventi in programma;
- monitoraggio dello stato di salute del dipendente tramite dispositivi wearable in caso di emergenze - per esempio in fabbriche con temperature particolari;
- integrazioni ad-hoc richieste dalle singole aziende.

Capitolo 6

Conclusioni

L'interesse verso i *wearable* è fortissimo in questo momento poichè con molta probabilità sarà il settore che darà, e che sta dando, nuovi stimoli ed ambiti di ricerca a tutta l'industria informatica.

Come descritto da Marie Chan e altri [6] sull'utilizzo dei wearable in medicina - anche se gli stessi concetti possono applicarsi in generale - ci sono sfide da accogliere e problematiche nel settore che dovranno essere affrontate nei prossimi anni. Tra queste troviamo:

- miglioramento delle caratteristiche hardware e delle prestazioni;
- efficienza, affidabilità e discrezionalità;
- garanzia della privacy degli utenti;
- legislazione e diritto, soprattutto per quanto riguarda l'ambito dei dati sanitari;
- interoperabilità e compatibilità con sistemi esistenti.

Il lavoro svolto è stato soddisfacente e altamente formante. Sono state gettate le fondamenta per questo progetto e per i futuri che riguarderanno Apple Watch e il mondo dei dispositivi *wearable*. Considerando che prima d'ora non mi ero ancora avvicinato al mondo della comunicazione via Bluetooth, è stata un'ottima occasione per studiare il protocollo e i framework a disposizione sulle piattaforme utilizzate.

È stato motivante lavorare su due piattaforme molto particolari e specifiche, quella di Apple Watch che è ancora agli albori (App Store possiede

attualmente 10.000 app), e quella del terminale, che con il collegamento della testina ha permesso nuovi tipi di interfacciamento su un dispositivo normalmente utilizzato solo per timbrare il proprio accesso tramite il badge.

Questo lavoro è servito per imparare nuovi concetti e perfezionare la conoscenza dei sistemi iOS e watchOS - oltre a Symfony, che già conoscevo ma che ancora non avevo avuto occasione di toccare con mano. Sicuramente è stato di molto stimolo il supporto totale e l'aiuto fin dall'inizio delle due aziende nella progettazione del prototipo e nella stesura della tesi, che hanno permesso fin da subito di superare tutti gli ostacoli e di capire bene il comportamento dell'hardware fornito, per poi scriverci sopra un software funzionante.

Appendice A

Concorrenza in iOS e OS X con Grand Central Dispatch

Un breve approfondimento sugli strumenti di gestione della concorrenza messi a disposizione da iOS e Mac OS X.

Grand Central Dispatch (GCD) è una tecnologia sviluppata da Apple per il supporto alla scrittura di codice ottimizzato su hardware multicore per OS X e iOS. Il codice sorgente di GCD è disponibile liberamente sotto licenza Apache con il nome di `libdispatch`. Il pattern di riferimento è il thread pool. È stato presentato per la prima volta alla WWDC 2009 ed è disponibile a partire da iOS 4 e Mac OS X 10.6 Snow Leopard.

GCD lavora tramite task inseriti in una coda di esecuzione gestita dai core del processore. Sotto l'aspetto dello sviluppo, GCD astrae il concetto di "thread" e libera il programmatore dalla gestione di aspetti di multithreading a basso livello. Nel codice, GCD è implementato tramite funzioni o blocchi, estensioni dei linguaggi C, C++, Objective-C e Swift che permettono di dichiarare funzioni inline.

Un semplice esempio di codice che fa uso di Grand Central Dispatch è il seguente:

Listing A.1: Esempio di operazione concorrente in GCD

```
1 dispatch_async(dispatch_get_global_queue(
    DISPATCH_QUEUE_PRIORITY_HIGH, 0), ^{
2     //do some things in background
3     //when I'm done, call:
4     dispatch_async(dispatch_get_main_queue(), ^{
```

```
5                 //update UI
6         })
7 })
```

In iOS è possibile utilizzare anche le classi `NSOperation` e `NSOperationQueue`. Tali classi prima di iOS 4 e Mac OS X 10.6 Snow Leopard avevano un funzionamento diverso rispetto a GCD; ora sono diventate dei semplici wrapper di GCD.

Per gestire invece i thread indipendentemente è possibile utilizzare `NSThread` ad alto livello e i thread POSIX (`pthread`) a basso livello.

Gestione della priorità dei task con le Classi di Qualità del Servizio

iOS 8 e Swift hanno introdotto in GCD le Classi di Qualità del Servizio (*Quality of Service Classes*)[1]. Le classi di QoS sono delle categorizzazioni del lavoro svolto dai thread. Assegnando una classe di QoS ad un task, il sistema esegue i task utente e di sistema organizzati in base alla propria priorità.

Queste classi sono:

User interactive: task istantanei con cui si interagisce con l'utente, ad esempio l'utilizzo dell'interfaccia o delle animazioni;

User initiated: task semi-istantanei avviati dall'utente che richiedono un risultato immediato;

Default: valore di default della classe, la classe di QoS lavora tra User initiated e Utility;

Utility: task che richiedono alcuni secondi per essere completati, ad esempio il download di un file;

Background: task di durata maggiore (minuti o ore) invisibili all'utente;

Unspecified: assenza di informazioni sulla classe di QoS del task.

[1] Apple Inc., *Building Responsive and Efficient Apps with GCD*. Apple Worldwide Developers Conference 2015, San Francisco (USA), 11 Giugno 2015. `https://developer.apple.com/videos/wwdc/2015/?id=718` (visitato il 14 Settembre 2015)

Le classi di qualità del servizio sono applicabili non solo a GCD ma anche alle classi `NSOperation`, `NSOperationQueue`, `NSThread` e ai thread POSIX (`pthread`).

In iOS 8 le classi QoS vanno a sostituire i precedenti tipi di Dispatch Queue Priority di GCD: main thread (user-interactive), high (user-initiated), default, low (utility), background.

Un esempio di codice che fa uso delle classi QoS è il seguente:

Listing A.2: Esempio di operazione concorrente in GCD con l'uso di classi QoS

```
1 var qos_attr = dispatch_queue_attr_make_with_qos_class(attr,
    QOS_CLASS_USER_INITIATED, 0);
2 var queue = dispatch_queue_create(‘‘com.company.queueName’’,
    qos_attr);
3 dispatch_async(queue, ^{
4         //do some things in background
5         //when I’m done, call:
6         dispatch_async(dispatch_get_main_queue(), ^{
7                 //update UI
8         })
9 })
```

Bibliografia

[1] K. Ashton, *That "Internet of Things" thing*. RFiD Journal, 22 Giugno 2009.

[2] L. Atzori, A. Iera, G. Morabito. *The Internet of Things: A survey*. Computer Networking, 31 Maggio 2010.

[3] M. Swan, *Sensor Mania! The Internet of Things, Wearable Computing, Objective Metrics and the Quantified Self 2.0*. Journal of Sensor and Actuator Networks, n°1, 2012, pp. 217-253.

[4] M. Weiser, R. Gold, J. S. Brown. *The origins of ubiquitous computing research at PARC in the late 1980s*. IBM Systems Journal, vol. 38, n° 4, 1999, pp. 693-696.

[5] T. Starner, J. Auxier, D. Ashbrook, M. Gandy, *The Gesture Pendant: A Self-illuminating, Wearable, Infrared Computer Vision System for Home Automation Control and Medical Monitoring*. The 4th International Symposium on Wearable Computers, Atlanta (USA), 16-17 Ottobre 2000.

[6] M. Chan, D. Estève, J. Fourniols, C. Escriba, E. Campo, *Smart wearable systems: Current status and future challenges*. Artificial Intelligence in Medicine, n° 56, 2012, pp. 137-156.

[7] Apple Inc., *WWDC 2015: The App Effect*. Apple Worldwide Developers Conference 2015, San Francisco (USA), 8 Giugno 2015. `http://www.macstories.net/news/the-numbers-from-apples-wwdc-2015-keynote/`. (visitato il 7 Agosto 2015)

[8] Zucchetti Axess, *Zucchetti Axess TMC 930 X1 & X2 Manuale Utente rev. 1.9*, Giugno 2013.

[9] Statista.com, *Global smartphone sales 2009-2014 by OS*. http://www.statista.com/statistics/263445/global-smartphone-sales-by-operating-system-since-2009/ (visitato il 8 Settembre 2015)

[10] Gartner, Inc., *Gartner's 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business*, 11 Agosto 2014. http://www.gartner.com/newsroom/id/2819918 (visitato il 12 Settembre 2015)

[11] Bluetooth SIG Inc., *Bluetooth Developer Portal*. http://developer.bluetooth.org (visitato il 16 Agosto 2015)

[12] Estimote, Inc., *Beacons technical specifications*. https://community.estimote.com/hc/en-us/articles/203159703-What-is-the-technical-specification-of-Estimote-Beacons (visitato il 18 Agosto 2015)

[13] Apple, Inc., *Core Location Framework Reference*. https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CoreLocation_Framework/ (visitato il 6 Settembre 2015)

Elenco delle figure

Listings

Ringraziamenti

Desidero ringraziare in primis la mia famiglia, la quale mi ha sostenuto ed aiutato durante questi tre anni di percorso universitario.

Ringrazio il professor Alessandro Ricci per l'ampia disponibilità dimostrata nel seguirmi in questo progetto di tesi e per i preziosissimi consigli dispensati.

Grazie ai miei colleghi di lavoro di Mr. APPs Giacomo, Samuele, Nicolò, Luca, Denis, Alessandro, Nicola, Matteo e Alessandra con cui ho potuto condividere questo percorso di alternanza studio-lavoro. Un ringraziamento particolare a Lorenzo che mi ha seguito come co-relatore durante questi ultimi mesi, ad Antonio Borsetti e a tutto il team di Zeitgroup che ci ha supportato e ha creduto fin dall'inizio al progetto.

Infine un ringraziamento ai miei amici e ai miei compagni di corso, in special modo a Aldo, Michele e Giacomo con i quali ho affrontato lezioni, giornate di studio, esami e progetti universitari. È stato molto bello trascorrere questi tre anni assieme a voi!