ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

KIBI – SOCIAL GAME INTERATTIVO IN AMBIENTE iOS

Relazione finale in

Mobile Web Design

Relatore

Dott. Mirko Ravaioli

Presentata da

Nicola Innocenti

Sessione III

Anno Accademico 2012 / 2013

INDICE

Introduzione

Ogni anno la tecnologia informatica si evolve sempre di più portando cambiamenti, migliorie e innovazioni.
Negli ultimi anni è andato affermandosi diventando, probabilmente, il mercato tecnologico di punta, il cosiddetto "smartphone", un dispositivo mobile che permette un utilizzo a 360 gradi da parte dell'utente di funzionalità che prima erano disponibili solo su computer.
L'affermazione di questi device si può collocare nel gennaio 2007 quando Steve Jobs (ex CEO della Apple) presentò alla conferenza di apertura del Macworld il primo iPhone, dopo la quale iniziò una vera e propria "corsa all'oro" da parte di tutte le maggiori società tecnologiche mondiali per produrre dispositivi equivalenti.
Uno *smartphone* offre abilità di computazione e di connettività molto maggiori rispetto a telefoni cellulari anche di ultima generazione, a differenza dei quali si avvalgono di un sistema operativo completo che fornisce una piattaforma per sviluppatori di applicazioni di terze parti.
Uno smartphone, dunque, combina assieme le funzioni di un PDA (*Personal Digital Assistant)* ed un telefono cellulare, che solitamente è un telefono provvisto di fotocamera e GPS. Per quanto riguarda la connettività utilizzano connessioni di tipo GSM, GPRS, EDGE, UMTS, HSDPA, HSUPA per non dimenticare Bluetooth e Wi-Fi.

Come precedentemente accennato a controllare i dispositivi mobili è un sistema operativo mobile, o mobile OS, così come un sistema operativo Windows, Mac OS o Linux controlla un computer desktop o portatile. I sistemi operativi mobile, però, sono molto più semplici sia perché devono lavorare su dispositivi con caratteristiche più limitate sia perché essendo sempre in movimento gli utenti necessitano di velocità di computazione e facilità di utilizzo.

Ma qual è il cavallo di battaglia di questi dispositivi? Le **app**.

Una app (termine derivato dall'abbreviazione di “applicazione”) è una variante dei software informatici come siamo abituati a pensarli su dispositivi fissi. Si differenzia dalle tradizionali applicazioni sia per il supporto su cui viene utilizzata (smartphone) sia per la concezione che racchiude in se.
E' caratterizzata da una semplificazione ed eliminazione del superfluo, al fine di ottenere leggerezza, essenzialità e velocità. Già il nome abbreviato può essere percepito come una semplificazione del nome completo per dare l'idea di qualcosa di piccolo e semplice.

E' molto importante per una app la sua chiarezza e facilità nei confronti dell'utente, il quale avrà una migliore esperienza d'uso velocizzando operazioni che prima era abituato a compiere più macchinosamente.
Una app può essere sviluppata per diversi sistemi operativi e non tutte sono compatibili con ogni tipo di sistema. Al fine di semplificare la ricerca e l'utilizzo delle applicazioni da parte di utenti anche inesperti, la loro distribuzione è gestita da appositi distributori digitali (conosciuti come *store* o *market*). Ogni tipo di distributore è vincolato ad un sistema operativo, affinché contenga al proprio interno solo applicazioni compatibili con il sistema operativo del dispositivo mobile che si sta utilizzando.

Lo sviluppo di app mobile era inizialmente destinato esclusivamente alla produttività aziendale e individuale: project managament, e-commerce, posta elettronica, ecc... Successivamente, complice la crescente domanda pubblica dovuta alla rapida diffusione dei moderni dispositivi mobili, è stata registrata la rapida espansione in altre aree, come ad esempio giochi, scienza applicata, automazione industriale, GPS.
Oggi esistono centinaia di migliaia di app che permettono all'utente un utilizzo a tutto tondo di abitudini quotidiane quali fare foto, giocare, leggere giornali, ottenere indicazioni stradali, prenotare biglietti on-line ma soprattutto condividere informazioni e dati tramite social network.

Capitolo 1

STATO DELL'ARTE

In questo capitolo verrà presentato il quadro generale nel quale si colloca l'applicazione sviluppata partendo da una rapida descrizione delle caratteristiche delle app e l'utilizzo che ne fa un utente fino ad arrivare ai maggiori sistemi operativi mobile attualmente sul mercato.
Successivamente si focalizzerà la descrizione sul device e il relativo sistema operativo per i quali è stata pensata questa app.
Una sezione è stata dedicata alla distribuzione digitale del software (gli store market) in quanto ricopre un ruolo fondamentale per la divulgazione e propagazione delle applicazioni.
Si passa poi ad una descrizione di alcune applicazioni già esistenti sul mercato che appartengono alla tipologia di app qui descritta.

1.1 Caratteristiche delle app

Ci sono tantissime tipologie di app da poter sviluppare, con svariati utilizzi. Inizialmente riassumerne una gerarchia era molto più semplice. Vi erano delle app "capostipite" di diverse aziende/società che offrivano gli stessi servizi. Basti pensare a giochi, meteo, news, foto, ecc...
Nel corso degli anni questa gerarchia è andata espandendosi portando tante novità nel modo di fare app. Nuovi modi di proporre features già esistenti sotto chiave molto più accattivante più comprensibile, più fantasiosa.
Un esempio possono essere le app di messaggistica istantanea, che nonostante offrano tutte lo stesso identico servizio (ovvero poter "chattare" con i propri contatti) lo presentano con varie caratteristiche quali :
- emoticons da aggiungere ai testi
- condivisione file, foto, audio, video

- videochiamate
- chat di gruppo
- chiamate vocali

Questo cosa significa? Che non è tanto il servizio a portare il successo di una applicazione quanto il modo in cui si pone. Se non ci fosse questo continuo cambiamento nello sviluppare applicazioni si cadrebbe in un continuo "già visto" facendo dominare solamente quelle app che ormai sono di dominio pubblico e utilizzate dalla maggior parte degli utenti.
Un'altra fondamentale caratteristica delle applicazione è l'interazione con l'utilizzatore. Gli smartphone sono dotati di vari sensori (GPS, accelerometro, giroscopio, ecc …) che ne migliorano l'uso o lo rendono più accattivante, scatenando certe azioni o eventi al verificarsi di una azione del possessore del telefono. Anche solo lo schermo touch è un esempio lampante. Lo stupore nel vedere accadere qualcosa nel toccare un oggetto particolare ha sicuramente un impatto diverso che a scorrere una lista con dei tasti freccia direzionali.

1.2 Utilizzo da parte dell'utente

Quando è sbocciato il periodo degli smartphone, e quindi delle app, l'utente medio faceva un utilizzo più longevo di queste ultime, colpito dalla novità principalmente, che lo portava a farne un utilizzo più costante.
Con la distribuzione di massa degli smartphone sono andate a crearsi tantissime sfaccettature di utilizzo delle app da parte degli utenti, come la moltitudine di sfumature di grigio che vanno dal nero al bianco.
E' importante valutare l'utilizzo dell'utente medio per poter sviluppare una applicazione che non lo porti ad abbandonarla in breve tempo. L'obiettivo finale di una app (guadagno escluso) è l'intrattenimento, l'utilità e l'esperienza d'uso. Nel caso di un gioco si punterà all'intrattenimento per convincere un giocatore a non scartarlo dopo qualche partita mentre, per esempio, in una applicazione di navigazione GPS si cercherà di disporre

l'utente di una interfaccia di facile comprendonio nonché di opzioni che rendono la, appunto, navigazione precisa e ricca di informazioni.

1.3 Scelta del sistema operativo

La crescente importanza dei dispositivi mobili ha messo in moto una forte competizione tra alcuni giganti del software come Google, Microsoft ed Apple insieme alle industrie leader nel settore mobile per cercare di catturare subito la fetta di mercato più ampia.

La scelta poteva ricadere quindi su uno tra questi sistemi operativi: Android (Google), iOS (Apple) e Windows Phone (Microsoft). Analizziamoli brevemente in ordine di anzianità.

IOS

Sistema operativo per dispositivi mobile sviluppato Apple. E' stato presentato il 9 gennaio 2007 al Macworld Conference & Expo, e la versione 1.0, ancora priva di nome, è entrata in commercio con il primo iPhone il 29 giugno dello stesso anno. L'11 luglio 2008 viene pubblicato in concomitanza della vendita di iPhone 3G l'aggiornamento ad iOS 2.0 che aggiunge, tra le altre funzioni, il molto atteso App Store e la possibilità di installare applicazioni di terze parti tramite l'app.
Come Mac OS X, è una derivazione di UNIX, tuttavia le applicazioni per MAC OS X non possono essere ufficialmente copiate e lanciate in dispositivi con iOS ma necessitano di essere modificate e compilate specificatamente per iOS.
A differenza di Android, iOS non è Open Source ed è un sistema molto chiuso, non lascia molte libertà di personalizzazione puntando principalmente sulle prestazioni.
iOS risulta un sistema operativo molto comodo nella programmazione in quanto praticamente tutti i device (tranne quelli molto datati) dispongono

dell'ultima versione facilitando senza dubbio il lavoro dello sviluppatore. Stesso discorso vale (per ora) per la grandezza degli schermi, solo 2.
Come ambiente di sviluppo si utilizza xCode, completo, di facile intuizione e compreso di emulatore molto rapido ed efficiente.
La praticità e le prestazioni di questo sistema operativo hanno permesso ad Apple di conquistare una grande parte del mercato mobile.

Android

Sistema operativo per dispositivi mobile basato sul kernel Linux sviluppato da Google. La presentazione del "robottino verde" avvenne il 5 novembre 2007 dalla neonata OHA (Open Handset Alliance), un consorzio di aziende del settore High Tech che include Google, produttori di smarphone come HTC e Samsung, operatori di telefonia mobile e produttori di microprocessori. Il primo dispositivo equipaggiato con Android che fu lanciato sul mercato fu l'HTC Dream il 22 ottobre del 2008.
Android si caratterizza per la struttura Open Source e il suo basarsi su kernel Linux. La caratteristica Open Source permette di modificare e distribuire liberamente il codice sorgente. Le applicazioni realizzate per Android sono scritte soprattutto con una versione modificata del linguaggio di programmazione java. L'enorme quantità di applicazioni presenti nello store e le caratteristiche descritte precedentemente hanno portato questo sistema operativo ad essere probabilmente la maggiore fetta di mercato mobile.
Il maggiore difetto di Android sono le troppe versioni del sistema operativo che viene principalmente aggiornato solo sugli smartphone più recenti. Ogni versione del sistema operativo porta cambiamenti nel codice apportando modifiche a metodi già esistenti o aggiungendo feature nuove. La maggior parte dell'utenza Android non dispone dell'ultima release del software e questo può risultare ostico nello sviluppo di una app.
Un altro "problema" sono le svariate dimensioni degli schermi degli smartphone con dotazione Android che costringono a disporre l'applicazione di svariati layout grafici.
Come ambiente di sviluppo si utilizza Eclipse o il nuovo Android Studio. Ho

avuto modo di testare solo Eclipse e posso affermare di non essermi trovato molto bene, né come ambiente né come emulatore.

Windows Phone

Sistema operativo per dispositivi mobile sviluppato da Microsoft presentato al Mobile World Congress il 15 febbraio 2010. Il nome deriva in parte dal vecchio OS di Microsoft chiamato Windows Mobile.
Se iOS e Android si somigliano in quanto ad interfaccia, Windows Phone si pone in chiave completamente diversa con delle "Live titles" che sono collegamenti ad applicazioni.
Come linguaggio di programmazione utilizza C#, Visual Basic o C++ mentre per le interfacce grafiche XAML.
L'ambiente di sviluppo è Visual Studio, un ambiente ricco di opzioni, facile e con un debug efficientissimo che risparmia molte fatiche ad uno sviluppatore.

La mia scelta del sistema operativo su cui appoggiarsi per lo sviluppo della app è ricaduta su iOS. Android avevo già avuto modo di utilizzarlo per lo sviluppo di una applicazione per l'esame di Mobile Web Design nel corso di laurea, avendo non pochi problemi, mentre Windows Phone sta cominciando a risalire il mercato dei dispositivi mobile solo ora perciò ho puntato sulla punta di diamante attuale in quanto a divulgazione e profitto.

1.4 iPhone e iOS

Riprendendo la rapida descrizione del paragrafo 2.2, dal 2007, anno di nascita del primo iPhone, si sono susseguite diverse generazioni di questo smartphone. L'ultima, attualmente, è l'iPhone 5S il quale ha mantenuto lo stesso identico design del suo predecessore, iPhone 5, con un display retina da 4 pollici con risoluzione 1136x640 pollici a 326ppi. La grande novità che ha portato questa versione al mondo degli smartphone è il suo processore

Apple A7 a 64 bit nonché il sistema operativo, iOS7, basato su di esso che permette di sfruttare al meglio le potenzialità di questo device da parte di applicazioni sviluppate con questo supporto.
Altra importante innovazione è il sensore di impronte digitali presente nel tasto home il quale permette una maggiore sicurezza per lo sblocco del device; viene inoltre sfruttato per la conferma dei pagamenti nell'App Store.

Una volta optato per il sistema operativo Apple, si è scelto di sviluppare l'app per iPhone escludendo i fratelli iPod Touch e iPad. Il primo è stato escluso in quanto presenta la mancanza del GPS, costringendo così la geolocalizzazione alla sola connessione Wi-Fi che non sempre è presente.
Relativamente all'iPad l'applicazione è facilmente trasferibile su questo dispositivo con opportune modifiche al progetto ma è stato escluso poiché l'applicazione è stata pensata in parte per l'utilizzo del GPS il quale comporta il doversi spostare con il device ed esso risulta troppo scomodo rispetto ad un iPhone.

1.5 Distribuzione digitale

La distribuzione digitale è la pratica di consegnare contenuti senza l'utilizzo di mezzi fisici, in particolare scaricandoli da internet direttamente sul dispositivo dell'utente.
In particolare nel nostro caso si tratta di software (applicazioni) per dispositivi mobili. Le principali piattaforme di distribuzione digitale che operano in questo campo sono, facendo riferendo ai tre maggiori sistemi operativi mobile, l'App Store della Apple, il Google Play Store di Google e il Windows Store di Microsoft.
Ciascun servizio oltre ad avere un software o un sito internet relativo è accessibile attraverso una applicazione pre-installata sul dispositivo permettendo di scaricare gratis o a pagamento migliaia di app pubblicate da terze parti.
Questo mercato sta subendo variazioni per quanto riguarda le applicazioni

gratuite. Le tipologie le possiamo semplificare in questo modo:

- App gratuita con fastidiosi banner pubblicitari e piccole feature non disponibili che spingono l'utente a comprare la versione a pagamento.
- App gratuita con tante limitazioni di utilizzo con acquisti cosiddetti *in-app*, all'interno della applicazione stessa che permettono di estenderne le feature.

In generale questo sistema di distribuzione digitale consente all'utente di avere a portata di mano, ovunque vada, tutte le applicazioni di cui ha bisogno semplicemente eseguendo qualche tap nello store potendo scegliere tra categorie, classifiche, novità, ecc...

Di seguito verrà fatta una breve panoramica sulla situazione e il funzionamento dell'App Store essendo la nostra applicazione sviluppata per iPhone.

App Store

L'app store è un servizio realizzato da Apple per iPhone, iPod Touch e iPad che permette agli utenti di acquistare e scaricare applicazioni disponibili in iTunes Store. Le applicazioni possono essere sia gratuite che a pagamento e possono essere scaricate direttamente dal dispositivo tramite l'apposita applicazione o dal computer.
L'App Store è stato aperto il 10 luglio 2008 tramite un aggiornamento software di iTunes, giorno nel quale Steve Jobs annunciò la presenza di 500 applicazioni di terze parti presenti in esso. Dal giorno seguente fu possibile per gli utenti scaricare le applicazioni tramite il proprio dispositivo raggiungendo nel giro di una settimana ad un totale di 10 milioni di download.
Il 23 aprile 2009 si raggiunse il miliardo di applicazioni scaricate.
Aspetto molto importante dello store è la "community". E' possibile valutare una applicazione con stelle che vanno da una a cinque e lasciare un commento. Queste due caratteristiche (presenti anche in altri store) sono

fondamentali per avere una idea della validità di un prodotto così come eventuali problemi per alcune versioni del sistema operativo o determinati device.
Utilizzando un account Apple, a cui è associata la carta di credito per il pagamento, si possono acquistare e scaricare le app direttamente nel proprio dispositivo. Le app di terze parti sono disinstallabili.
Il 20 ottobre 2010, invece, è stato annunciato il rilascio del Mac App Store, identico all'App Store dei dispositivi mobile ma inerente agli applicativi per Mac OS X quindi per mac fissi e macbook.
Questo fa capire quanto sia importante la vendita di prodotti digitali, laddove non è necessario un supporto fisico, da parte di Apple.

L'applicazione descritta in questo documento non è presente nello store in quanto Apple permette l'upload di applicazioni di terze parti solo qualora si sia in possesso di una licenza al costo di una quota annuale.
Risulta comunque molto interessante questo sistema di vendita in quanto permette agli sviluppatori di rendere disponibile la propria applicazione a migliaia di utenti in tutto il mondo senza intermediari e con pochi semplici passi.

1.6 Applicazioni simili già esistenti negli Store

Per quanto concerne la tipologia di app, la mia intenzione è stata di prenderla come "scusante" per imparare a programmare utilizzando i sensori dell'iPhone perciò mi sono accordato con il mio professore referente Dott. Mirko Ravaioli per lo sviluppo di un Tamagotchi.

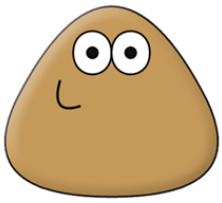
Il Tamagotchi è nato come gioco elettronico portatile nel 1996 da Aki Maita e prodotto dalla Bandai. L'obiettivo del gioco era quello di prendersi cura di una specie aliena (chiamata appunto Tamagotchi) e dargli il necessario per farlo crescere, facendolo vivere il più a lungo possibile e curandolo in caso di malattia.

Dal 1997 al 2006 sono state create ben 37 versioni portatili del Tamagotchi nonché alcune versioni videoludiche ispirate al gioco originale.
Ovviamente non mancano giochi mobile ispirati al capostipite Tamagotchi, con soggetti di cui prendersi cura molto diversi tra loro.
Negli store applicazioni di questo tipo ce ne sono diverse che ricalcano lo stile del Tamagotchi, ne fornirò due come esempio.

Clumsy Ninja

Questa applicazione è un Tamagotchi nel quale ci si deve prendere cura di un ninja completamente in 3d e con una interazione col personaggio davvero ben fatta.
A differenza di tante altre applicazioni di questo tipo, Clumsy Ninja ha una "trama". Lo scopo del gioco è allenare il piccolo, e alquanto maldestro, ninja per ritrovare la sua amica scomparsa.
Durante il gioco è possibile fargli fare svariati allenamenti quali saltare su tappeti elastici e dare pugni al sacco da boxe.

Oltre a tutto questo ci sarà il sensei, l'allenatore vero e proprio che noi aiuteremo, che ci assegnerà degli incarichi da compiere per guadagnare punti e livelli. Salendo di livello sarà possibile ottenere nuove cinture da far indossare al ninja nonché la possibilità di comprare nuovi indumenti.



Può

A differenza di Clumsy Ninja, Pou è un Tamagotchi molto più classico. Ci si dovrà prendere cura di quella che, apparentemente, sembra una patata. Lo stile di gioco è molto semplice, ci sono diverse stanze, intercambiabili attraverso appositi bottoni, nelle quali si potrà interagire con Pou in maniera differente. Ad esempio nel bagno lo si potrà lavare cospargendolo di schiuma e togliendola con l'acqua della doccia. In cucina gli si potrà dare da mangiare, ecc...
L'aspetto novità di questa applicazione è il reparto "giochi", nel quale si potrà far divertire il nostro personaggio tramite tanti piccoli giochi come se

fossero ciascuno una mini applicazione.

La grande interazione raggiunta da queste applicazioni nei confronti dell'utente è senza dubbio la notifica locale per avvisarlo che i suoi protetti hanno bisogno di qualcosa (un tantino invadente in certi casi).

Capitolo 2

PROGETTAZIONE

In questo capitolo verrà descritto il progetto e le metodologie adottate proseguendo successivamente con i requisiti, la struttura con alcuni screenshot del funzionamento dell'applicazione, la presentazione delle principali problematiche sorte durante il processo di progettazione e gli approcci alla loro risoluzione.

2.1 Scelta e design del soggetto

L'idea di sviluppare un Tamagotchi è nata in parallelo al soggetto stesso di cui prendersi cura, ovvero una fantomatica morosa.
Il nome della protagonista, Kibi, nasce dal vocabolo giapponese “chibi” che significa “persona bassa” o “bambino piccolo”. E' una parola molto utilizzata nel mondo dei manga e anime.
Avendo l'hobby di ricopiare personaggi di fumetti giapponesi sono partito subito con l'idea di realizzarla in quella chiave. A quel punto l'unico dubbio era sullo stile nel quale realizzarla... Matura? Realistica? Infantile? La scelta è ricaduta su “infantile” per un giudizio personale; credo che sfruttato bene un personaggio fumettistico disegnato così possa essere più divertente in quanto ad espressioni e animazioni.
Scelto come realizzare la protagonista è stata la volta della ricerca di un soggetto simile da cui prendere spunto per il disegno. Grazie alla ricerca immagini di Google sono riuscito a trovare in pochi minuti quello che cercavo, per riproporlo poi, utilizzando una tavoletta grafica, su Adobe Photoshop CS6.

2.2 Stile di gioco e scopo finale

Una volta al giorno verranno assegnati da Kibi cinque incarichi che si ha tempo di concludere entro le 12 del giorno successivo. Come “giorno” si intendono tutti quelli nei quali viene aperta l'applicazione per evitare la frustrazione di aprirla dopo qualche giorno e trovare il proprio punteggio al minimo. Ciascun incarico, una volta completato, assegnerà dei punti, alternativamente se allo scadere del giorno non è ancora stato portato a termine detrarrà punti. Completare tutti e cinque gli incarichi assegnerà punti bonus. Inoltre alla prima accensione giornaliera della applicazione ci sarà la possibilità di ottenere un incarico bonus che ha la peculiarità di non detrarre punti in caso di fallimento.
Gli incarichi sfrutteranno l'utilizzo di GPS, accelerometro, fotocamera, sensore di prossimità e giroscopio. Ad esempio un incarico potrebbe essere di avvicinarsi entro una determinata quantità di metri da un centro ospedaliero.
Il punteggio totale va da 0 a 100. Lo scopo del gioco è raggiungere i 100 punti, dopodiché si avrà spostato Kibi e si potrà condividere il proprio risultato nei maggiori social network.

Il gioco è impostato in modo tale da non diventare un “impegno” per il giocatore ed essere il meno invasivo possibile.
Solitamente un tamagotchi tende ad utilizzare notifiche locali per avvisare l'utente, diverse volte al giorno, che la sua creatura ha bisogno di attenzioni.
Questi continui avvisi li considero fastidiosi a lungo termine perciò ho scelto di utilizzare la notifica solamente come avviso della disponibilità di nuovi incarichi il giorno successivo alle 12 rispetto all'ultimo login effettuato.

2.3 Interfaccia grafica

A seguito del tocco sull'icona dell'applicazione verrà presentato per un paio di secondi uno splash screen.

Lo scopo dello splash screen è di mostrare all'utente il logo della applicazione mentre in background vengono fatti i caricamenti necessari.

In questo caso, lo splash screen è una view mostrata al di sopra di quella che è la vera view principale della applicazione. In realtà, mentre lo splash è visibile (per un paio di secondi), sotto di esso la view principale è già stata caricata.
L'immagine di sfondo dello splash screen è stata realizzata utilizzando Photoshop

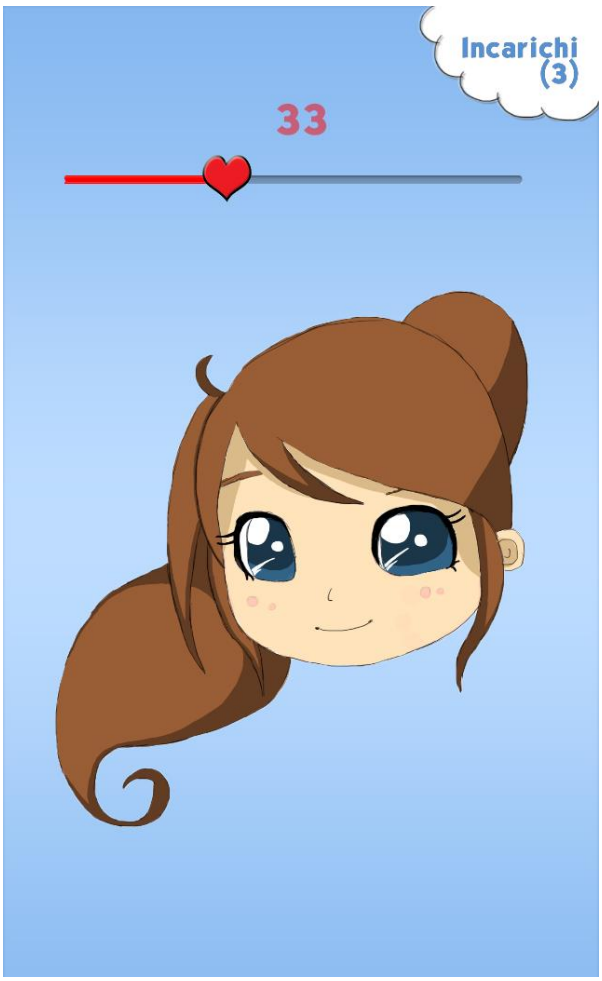
Al primo avvio della applicazione, però, invece della view principale verrà prima mostrata una ScrollView a scopo di “tutorial”.

La ScrollView è formata da cinque tab contenenti ciascuno una immagine che mostra, in maniera molto semplice, come funziona l'applicazione nelle sue principali caratteristiche. Il tutorial si può concludere con un apposito bottone che compare ai piedi della View una volta giunti all'ultima pagina, dopodiché non sarà più possibile visualizzarlo.

E' stato scelto di non dare la possibilità all'utente di visualizzare nuovamente il tutorial in quanto l'applicazione si presenta molto facile all'utilizzo e non necessita di spiegazioni particolari; gli incarichi stessi descrivono cosa si debba fare per concluderli.

Una volta concluso il tutorial o, in tutti gli altri casi di utilizzo dell'app,

viene visualizzata la view principale che si presenta in questa maniera.

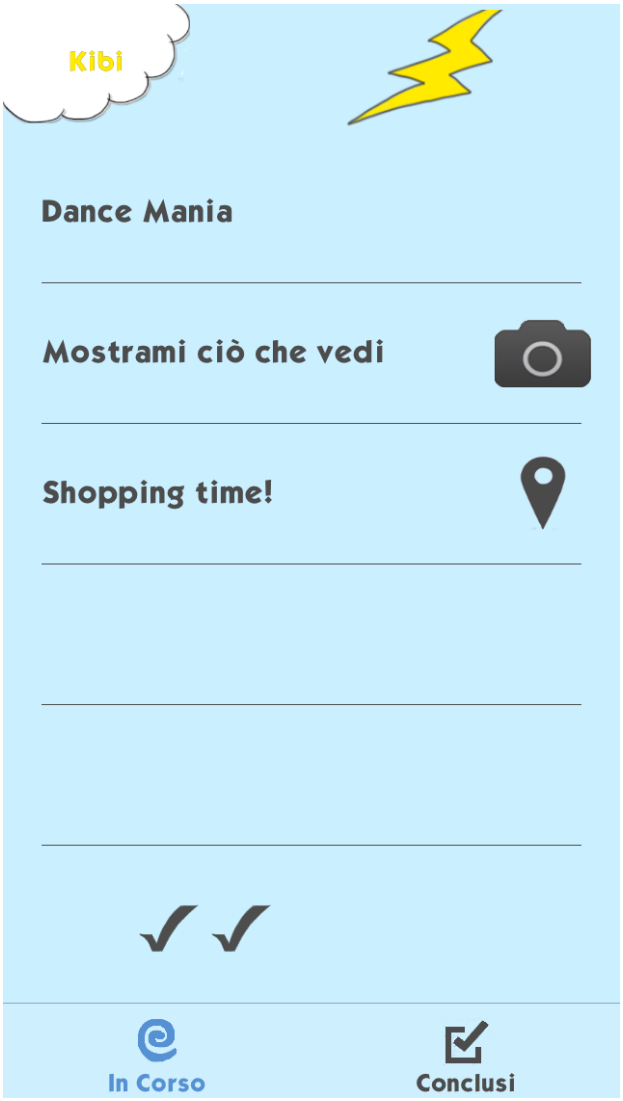


L'interfaccia è formata da:

- UIButton necessario per aprire la view degli incarichi. L'immagine di sfondo è una nuvola disegnata in Photoshop. La Label all'interno è aggiornata notificando all'utente quanti incarichi in corso ha attualmente.
- UISlider relativo al punteggio corrente dell'utente. E' uno slider personalizzato, preparato interamente in Photoshop. Per notificare il punteggio all'utente vi è una Label associata che ne riporta il valore.
- UIImageView nella quale troviamo la protagonista del gioco. Le immagini che appaiono in questa UIImageView sono tutti file .png disegnati in Photoshop.

Ci sono altre caratteristiche riguardanti l'interfaccia che, però, non sono statiche. Ad esempio, all'ottenimento di nuovi incarichi giornalieri comparirà una ulteriore UIImageView necessaria per una particolare animazione oltre ad una UILabel presente sotto l'immagine di Kibi con lo scopo di notificare quanti punti si è persi nell'ultima giornata di gioco.

Al click del bottone degli incarichi si aprirà una view gestita da un UITabBarController che, in base a quale tab si ha selezionato, apre una determinata view associata ad esso. Il tab aperto di default è quello relativo agli incarichi in corso.



L'interfaccia è formata da:

- UIButton simmetrico a quello presente nella view principale al cui click permette, per l'appunto, di tornare alla view principale
- UITableView contenente gli incarichi in corso
- UIImageView sottostante alla UITableView raffigurante il numero di incarichi completati in giornata tramite apposite “v”.
- UIImageView raffigurante un fulmine. Fa la sua comparsa solamente in caso sia attivo l'incarico speciale; associato ad essa può comparire un UIButton qualora sia necessario l'utilizzo della fotocamera o del GPS.

Le icone riguardanti il GPS e la fotocamera sono state realizzate in Photoshop a partire da immagini ottenute in rete per riprendere quella che è la simbologia più utilizzata al giorno d'oggi.
Per ottenere informazioni riguardanti il singolo incarico è necessario un click su uno di essi permettendo la comparsa di una view soprastante alla lista contenente i dettagli richiesti.

La view si presenta come una finestra. Contiene il titolo dell'incarico, la descrizione riguardante cosa è necessario fare per completarlo e i punti ottenibili al completamento. La UIImageView con l'immagine a “x” permette la chiusura della view.

Il tab riguardante gli incarichi completati, invece, si presenta in questa maniera:

L'interfaccia grafica è molto più semplice rispetto agli incarichi in corso ed è formata da:

- UIButton che permette di tornare alla view principale.
- UITableView contenente lo storico degli incarichi completati o falliti in tutti i giorni di gioco

Lo storico è raggruppato e ordinato per data. Per ciascun incarico sono definiti il titolo, il punteggio ottenuto o perso e il punteggio totale raggiunto dall'utente a seguito del fallimento/completamento dell'incarico. Come nel tab degli incarichi in corso è possibile visualizzare una finestra con i dettagli dell'incarico effettuando un click su quest'ultimo.

2.4 Problematiche progettuali

Il *primo aspetto* su cui si è riflettuto per la realizzazione di questa applicazione è stato l'interazione dell'utente con Kibi. E' stata valutata la possibilità di lasciare dei processi in background anche ad applicazione chiusa, in modo tale da notificare, saltuariamente, di effettuare determinate azioni riaprendo l'app.
Questa possibilità è stata scartata in quanto iOS è molto fiscale per quanto riguarda i servizi in background, permettendo la loro permanenza nel tempo solo nel caso, ad esempio, comunichino costantemente in rete o utilizzino il lettore musicale.
In definitiva si è scelto di impostare l'interazione utente-Kibi ad una sola notifica il giorno successivo all'ultima apertura dell'app per avvisare della possibilità di ottenere nuovi incarichi. In questa maniera l'applicazione è stata resa più gestionale che interattiva lasciando libero arbitrio all'utente di controllare a piacimento che incarichi può portare a termine.

Secondo aspetto considerato è stato la gestione del database contenente gli incarichi. Database online o intrinseco all'applicazione? Non trattandosi di un database contenente centinaia di tuple la scelta è ricaduta sul salvarlo internamente all'app.

Terzo aspetto è stato il modo nel quale l'utente avrebbe dovuto effettuare il check-in nelle zone richieste dalla app tramite GPS. L'idea iniziale era che, una volta essersi avvicinato ad un luogo che rispettava la categoria richiesta, veniva notificato all'utente il raggiungimento della destinazione. Questa

dinamica, però, richiede un sevizio in background che continui a monitorare la posizione tramite GPS, cosa che (come detto sopra) iOS non permette di fare se non in determinati casi.
La scelta dunque è ricaduta sul posizionare un bottone affianco agli incarichi richiedenti GPS, in modo tale da far fare il check-in manualmente all'utente.

Queste sono state le uniche problematiche riguardanti la progettazione, per il resto si è trattato di rendere l'interfaccia grafica il più facile possibile nei confronti dell'utente.

Capitolo 3

IMPLEMENTAZIONE

Qui di seguito verrà descritta l'implementazione della applicazione a partire dagli strumenti utilizzati con una breve descrizione. Nella parte finale sono elencate le problematiche riscontrate durante l'implementazione del progetto e il metodo che è stato scelto per la loro risoluzione.

3.1 Ambiente di sviluppo

Xcode

Xcode è un ambiente di sviluppo integrato (*Integrated Development Environment*, IDE) sviluppato da Apple Inc. per agevolare lo sviluppo di software per Mac OS X e iOS. E' fornito gratuitamente in bundle con il sistema operativo a partire da Mac OS X 10.3 Panther, sebbene sia in grado di generare programmi per qualsiasi versione di Mac OS X. Estende e rimpiazza il precedente tool di sviluppo della Apple, *Project Builder*, che era stato ereditato dalla NeXT. Ufficialmente Xcode non funziona su Mac OS X 10.2 Jaguar.

Xcode lavora in congiunzione con Interface Builder (anch'esso proveniente da NeXT) che permettono agli sviluppatori che usano Carbon e Cocoa di disegnare interfacce grafiche per le applicazioni usando uno strumento grafico, senza la necessità di scrivere decine di righe di codice. L'interfaccia risultante è salvata in un file .nib o, nelle versioni più recenti, in file .xib.

Xcode include GCC, che è in grado di compilare codice C, C++, Objective

C/C++ e Java. Supporta ovviamente i framework Carbon e Cocoa oltre a tanti altri.
Una delle caratteristiche più avanzate di Xcode è il supporto della distribuzione in rete del lavoro di compilazione. Usando Bonjour e Xgrid è in grado di compilare un progetto su più computer riducendo i tempi.

Supporta, inoltre, la compilazione incrementale, ovvero è in grado di compilare il codice mentre viene scritto.

Dalla versione 3.1, Xcode è anche lo strumento per sviluppare le applicazioni native per iPhone e iPod Touch. Dalla 3.2 si aggiunge lo sviluppo per iPad.
Nel marzo del 2011 è stata rilasciata la versione 4.0 la cui caratteristica più importante è la nuova interfaccia grafica che integra tutte le funzionalità all'interno di un'unica finestra, compreso l'Interface Builder. Altra novità introdotta da questa versione è la funzionalità Fixit che fornisce alcuni suggerimenti per il completamento del codice e l'individuazione dei più comuni bug o errori di battitura sottolineandoli con una linea rossa.
Infine Xcode 4 è stato reso disponibile a pagamento anche tramite la distribuzione digitale attraverso il Mac App Store.

Nel 2013 Xcode riceve un ulteriore importante aggiornamento, Xcode 5.
Con questa versione, oltre ad un leggero cambiamento all'interfaccia, gli strumenti forniti dall'IDE aiutano più che mai lo sviluppatore nella realizzazione rapida ed efficiente di applicazioni di alta qualità. Automaticamente Xcode configura le app per l'uso degli ultimi servizi Apple nonché la compatibilità con iOS7, ultima versione del sistema operativo mobile, e Mavericks, ultima versione del sistema operativo MAC OS X.
Per la realizzazione della applicazione descritta in questo documento mi sono avvalso di Xcode 5 perciò elencherò tre importanti features presenti in questa versione:
- Abilitazione di servizi Apple come iCloud, Passbook e Game Center direttamente all'interno dell'IDE.

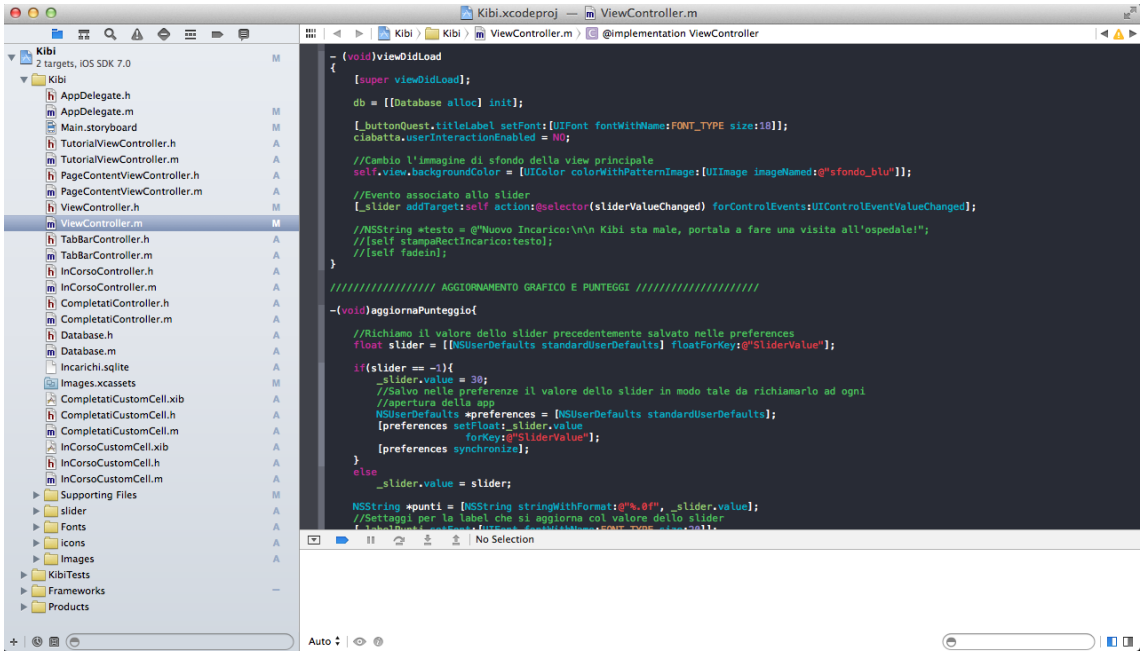
- Auto Layout dell'interfaccia utente. E' una potente tecnologia che permette di creare una interfaccia che si adatta automaticamente alla grandezza e orientamento dello schermo.
- L'Asset Catalog permette di gestire le immagini in maniera molto semplice. Si può lavorare con tutte le versioni di ciascuna immagine (Mac, iPhone, iPad) senza gestire i singoli file. Permette di scalare le immagini dinamicamente con il layout dell'interfaccia mantenendone piccola la dimensione del file.

Il funzionamento dell'ambiente di lavoro è molto intuitivo. Una volta creato un nuovo progetto e scelta la tipologia tra i template disponibili, appare l'area di lavoro.

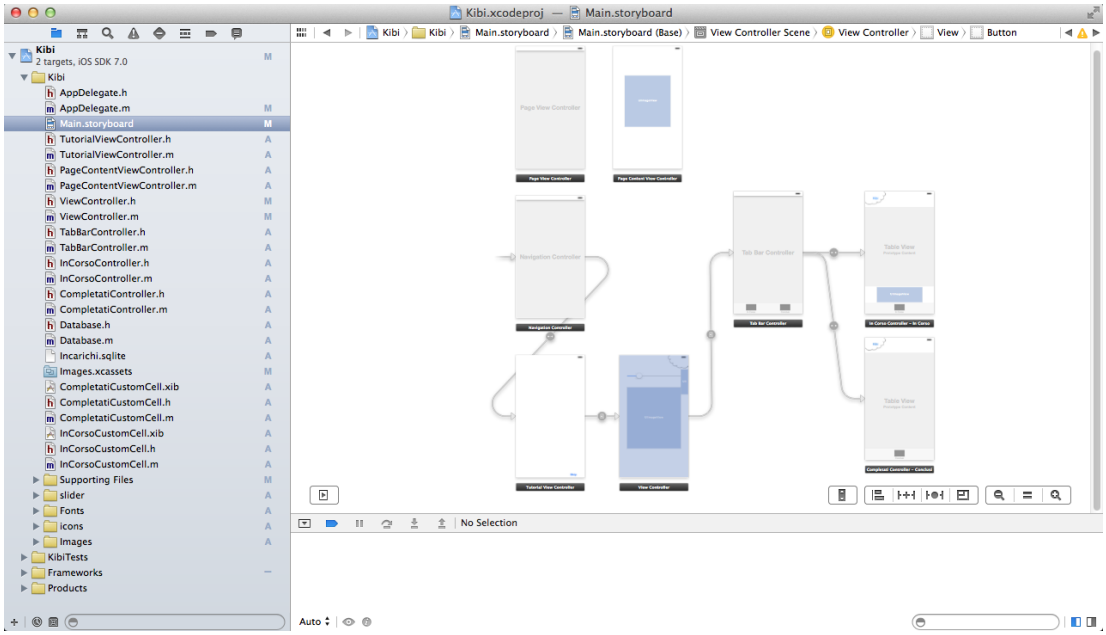
Nella colonna di sinistra compare la lista dei file, all'interno di cartelle, che compongono il progetto. In particolare è possibile creare interfacce sia per iPhone che per iPad e associargli specifiche classi. Nel mio caso ho sviluppato l'applicazione interamente per iPhone per cui tutte le classi sono relative ad esso.

Ho inoltre creato 2 cartelle contenenti rispettivamente le icone e le immagini utilizzate all'interno dell'app.

Ambiente di lavoro Xcode



Interface Builder



3.2 Database

Sqlite

SQLite è una libreria software scritta in linguaggio C che implementa un DBMS SQL di tipo ACID incorporabile all'interno di applicazioni.
Il suo creatore, *D. Richard Hipp*, lo ha rilasciato nel pubblico dominio, rendendolo utilizzabile quindi senza alcuna restrizione.
Permette di creare una base di dati (comprese tabelle, query, form, report) incorporata in un unico file, come nel caso dei moduli Access di Microsoft Office e base di OpenOffice.org.

SQLite non è un processo standalone utilizzabile di per sé, ma può essere incorporato all'interno di un altro programma. E' utilizzabile con il linguaggio C/C++, ed esistono binding anche per altri linguaggi, in particolare Tcl. E' inoltre stato integrato nella versione 5 di PHP, consentendo a tale popolare linguaggio di disporre di un altro RDBMS

indipendentemente dalla presenza di MySQL.

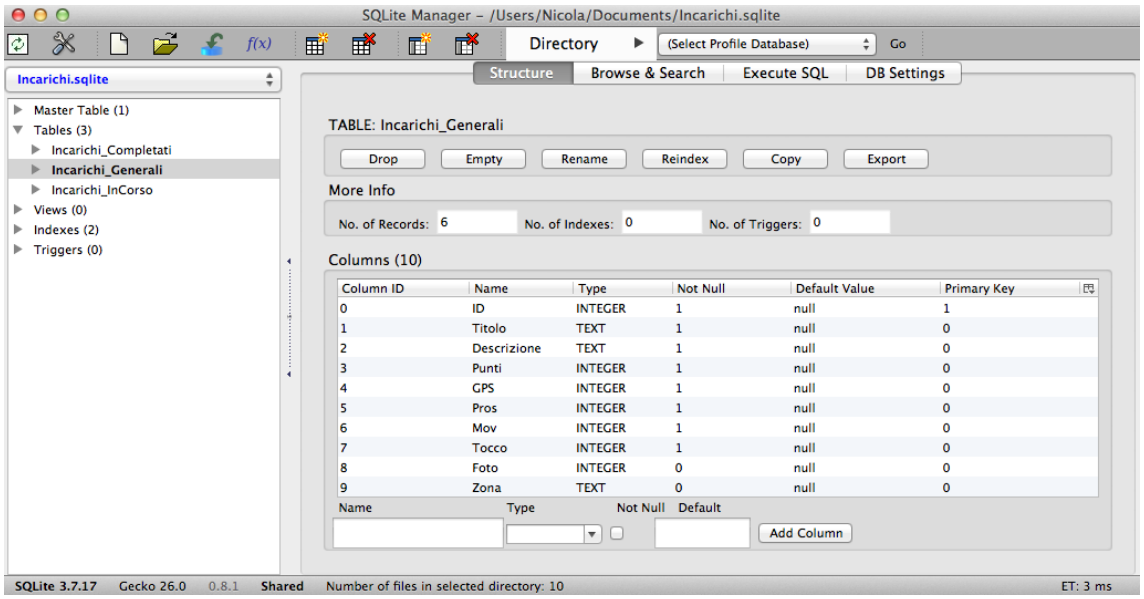
Viene utilizzato nel browser *Mozilla Firefox* e *Seamonkey* per memorizzare i bookmark, la cronologia di navigazione ed altre informazioni.

La libreria offre molte interessanti caratteristiche:

- è compatta (meno di 500KB per l'intera libreria alla versione 3.6.14)
- è molto veloce, in molti casi anche più di MySQL
- il codice sorgente è liberamente disponibile, chiaro e ben commentato
- è in grado di interpretare stringhe SQL
- l'API è semplice da utilizzare
- è multipiattaforma
- include un programma di utilità a riga di comando per accedere al database anche manualmente o tramite scripting
- supporta database che possono essere anche molto grandi, attualmente il limite è 2TB
- un database consiste di un unico file, il cui formato interno è indipendente dalla piattaforma e dal relativo ordine dei byte

Per la creazione del database ho sfruttato un componente aggiuntivo di Mozilla Firefox chiamato SQLite Manager che mi ha permesso di creare il file .sqlite del database aggiungendo tabelle, campi e i dati stessi con una interfaccia utente molto semplice e intuitiva.

Interfaccia SQLite Manager



3.3 Editor grafico per il personaggio e le animazioni

Adobe Photoshop CS6

Adobe Photoshop è un software proprietario prodotto dalla *Adobe Systems Incorporated* specializzato nell'elaborazione di fotografie e, più in generale, di immagini digitali.

Questo programma è in grado di effettuare ritocchi di qualità professionale alle immagini, offrendo enormi possibilità creative grazie ai numerosi filtri e strumenti che permettono di emulare le tecniche utilizzate nei laboratori fotografici per il processamento delle immagini, le tecniche di pittura e di disegno.

Un'importante funzione del programma è data dalla possibilità di lavorare con più “livelli”, permettendo di gestire separatamente le diverse componenti che costituiscono l'immagine principale.

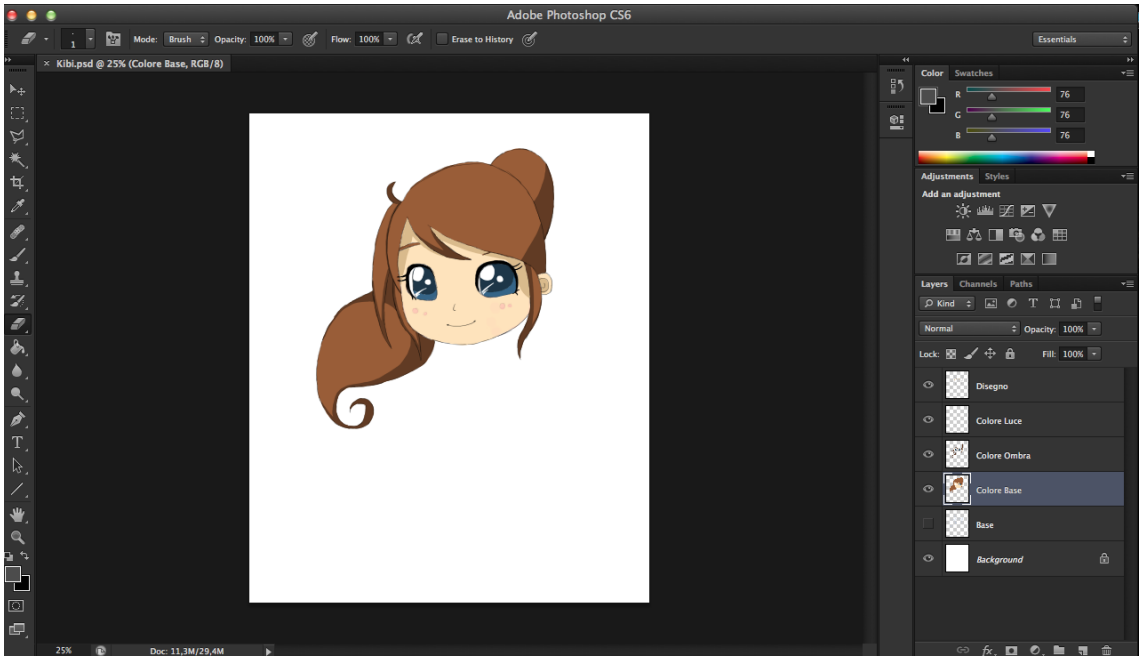
E' un software di grafica *raster* e in quanto tale si differenzia dai software di grafica *vettoriale* poiché agisce direttamente sui pixel, sebbene alcuni strumenti abbiano la possibilità di applicare gli algoritmi di Bézier.

Il software è espandibile mediante l'uso dei plug-in che permettono di ampliarne le funzionalità.

Adobe ha reso disponibili le specifiche del formato per la compilazione dei plug-in anche ad aziende di terze parti, creando quindi un mercato specifico di componenti per Photoshop, compresi anche pennelli, campioni di colore, pattern e sfumature, nonché di macro (comunemente chiamate “azioni”).

Il PSD è un formato di file grafico di Adobe Photoshop, in grado di salvare una immagine completa di tutti i livelli che la compongono. Una immagine salvata in formato proprietario Adobe ha il vantaggio di essere lavorabile in fasi successive. Lo svantaggio è dato dalle dimensioni eccessive che non rendono agevole il suo trasferimento, se non usando idonee connessioni veloci o supporti adeguati.

Interfaccia Adobe Photoshop CS6



3.4 Principali librerie utilizzate

CoreMotion

Il framework CoreMotion permette all'applicazione di ricevere dati relativi al movimento dall'hardware dello smartphone per poterli successivamente utilizzare.
Il framework supporta l'accesso ai dati dell'accelerometro sia grezzi che elaborati utilizzando interfacce basate su blocchi. La stessa cosa vale per quanto riguarda i dati del giroscopio, qualora sia incorporato nel device.
Si possono utilizzare i dati sia dell'accelerometro che del giroscopio per giochi o altre applicazioni che utilizzano il movimento come input o semplicemente come modo per migliorare l'esperienza d'uso da parte dell'utente.

In particolare questa libreria è stata utilizzata in nell'applicazione per simulare un ipotetico ballo muovendo il device.

CoreLocation

Il framework CoreLocation consente di determinare, tramite l'hardware disponibile nel device, la posizione corrente. Si possono utilizzare classi e protocolli di questo framework per configurare e programmare la consegna di particolari eventi basati sulla posizione.
Si può utilizzare anche per definire regioni geografiche e monitorare quando l'utente le attraversa. In iOS è possibile anche definire delle regioni attorno ad un beacon Bluetooth.

In particolare questa libreria è stata utilizzata nell'applicazione per gestire il check-in in determinati incarichi che richiedono di avvicinarsi entro 100 metri da un particolare luogo (es. ospedali, centri commerciali, musei, ecc).

Facebook SDK

API per integrare Facebook ad una applicazione. Le possibilità date da questa libreria sono tante quali:
- Registrarsi all'applicazione tramite i dati del profilo Facebook
- Ottenere i dati dal profilo facebook così come la lista degli amici
- Condividere contenuto nel proprio diario o in quelli dei contatti
- Creare notifiche sotto forma di storia (es. Carlo *ha ottenuto un obiettivo*)

In particolare questa libreria è stata utilizzata nell'applicazione per condividere sul diario del profilo Facebook dell'utente il punteggio ottenuto una volta concluso il gioco.

Twitter SDK

Così come per Facebook questa libreria permette di ottenere, dall'account Twitter dell'utente, tutti i contatti che si seguono, i propri dati personali, i follower e la possibilità di scrivere un “tweet”.

In particolare questa libreria è stata utilizzata nell'applicazione per condividere un tweet dal proprio profilo Twitter associato agli account dell'iPhone.

3.5 Principali classi dell'applicazione

ViewController

Classe che gestisce la view principale dell'intera applicazione, ovvero quella dove si trova il volto di Kibi.
Per quanto riguarda l'interfaccia grafica gestisce:

- UIImageView nel quale viene rappresentata Kibi. L'immagine presente cambia in base ad azioni dell'utente o a causa dello scatenarsi di particolari eventi;
- UISlider il cui scopo è segnalare, in parallelo ad una UILabel, il punteggio corrente ottenuto dall'utente;
- UIButton che permette di cambiare view e passare a quella relativa gli incarichi;
- UIImageView nascosta utilizzata unicamente per una particolare animazione che unisce il bottone degli incarichi a Kibi tramite piccole nuvole.

Per quanto riguarda il codice vero e proprio questa classe gestisce diversi aspetti.
Essendo la prima view ad apparire nell'applicazione la prima cosa che fa è un check della data in modo tale da assegnare cinque nuovi incarichi nel caso sia passato il mezzogiorno del giorno successivo all'ultimo nel quale ne abbiamo ricevuti.
Per assegnare nuovi incarichi vengono prima spostati quelli non portati a compimento nel giorno appena trascorso nella tabella “Incarichi_Completati” del database, con l'apposito flag di fallimento.

Liberata la tabella degli incarichi attualmente in corso, vengono scelti casualmente cinque nuovi incarichi tra tutti quelli disponibili che la andranno a riempire nuovamente. Inoltre c'è un 25% di possibilità che venga assegnato un sesto incarico speciale che, a differenza degli altri, in caso di fallimento non detrarrà punti.
In background ci sono invece due metodi che controllano, rispettivamente, uno il valore presente nello UISlider del punteggio per modificare, in relazione ad esso, l'immagine di Kibi, il secondo se ci sono incarichi in corso che richiedono sensori di prossimità o movimento e attivandoli in caso positivo.
In caso venga completato un incarico che richiede di essere in questa particolare view, viene gestita la sua conclusione spostandolo (come nel caso di fallimento) nella tabella degli incarichi completi col flag di successo attivo.
Il punteggio raggiunto dall'utente viene salvato negli NSUserDefaults, al cui interno si possono definire delle variabili di tipo chiave-valore che mantengono il loro contenuto anche al chiudersi dell'applicazione.

Esempio di creazione negli NSUserDefaults

```
NSUserDefaults *preferences = [NSUserDefaults standardUserDefaults];
[preferences setFloat:_slider.value
               forKey:@"SliderValue"];
[preferences synchronize];
```

Esempio di lettura dagli NSUserDefaults

```
float slider = [[NSUserDefaults standardUserDefaults] floatForKey:@"SliderValue"];
```

Come ultima caratteristica, nel caso si raggiunga il punteggio massimo (100), viene visualizzata una immagine con i bottoni di condivisione per Facebook e Twitter.

InCorsoController

Classe che gestisce una UITableView contenente la lista degli incarichi in corso. Graficamente troviamo:

- UIButton che permette di ritornare alla view principale;
- UITableView, lista degli incarichi in corso;
- UIImageView nascosta relativa all'incarico speciale, visualizzata solo in caso lo si sia ottenuto;
- UIButton nascosto relativo all'incarico speciale, qualora ne richieda l'utilizzo.

La chiave base di questa classe è di leggere dal database gli incarichi in corso, caricarli nella UITableView e gestirne il completamento.
Alcuni incarichi necessitano l'utilizzo di bottoni aggiuntivi nella relativa cella della lista. Ci sono due tipi di bottone interagibili con l'utente.

Il primo gestisce il check-in per la verifica del luogo in cui ci si trova. Al click del bottone viene attivato il Location Manager che fa un update del luogo in cui si trova l'utente salvandone le coordinate. Ottenute le coordinate, una richiesta URL al servizio di ricerca di Google restituisce le località specificate che si trovano vicino a noi, specificando nell'URL la tipologia di luogo, le coordinate in cui ci si trova e il raggio, in metri, della distanza massima da controllare.
Ottenuti i risultati dalla richiesta, si tiene conto del luogo più vicino a quello nel quale ci si trova e si restituisce all'utente procedendo con il completamento dell'incarico.
Per l'utilizzo del servizio Google è stato necessario attivare il servizio “Place Search” dalle API di Google stessa creando un riferimento alla applicazione e utilizzando una chiave associata ad esso che permetta la restituzione di dati alla richiesta URL.

Ecco un esempio di stringa contenente l'URL di richiesta.

```
richiesta = [NSString stringWithFormat:@"https://maps.googleapis.com/maps/api/place/search/json?location=%f,
    %f&radius=2000&types=%@&sensor=false&key=%@", newLocation.coordinate.latitude, newLocation.coordinate.longitude, [[incarichi
    objectAtIndex:indiceIncarico] objectForKey:@"Zona"], key];
```

Utilizzando il metodo “stringWithFormat” è possibile allegare ad una stringa del contenuto non di tipo testo. In questo caso è stato necessario allegare

latitudine, longitudine, la tipologia di luogo da ricercare (viene qui recuperata dal relativo campo nell'array degli incarichi in corso) e la chiave assegnata all'applicazione nella sezione delle API di Google per permettere la richiesta.
Il risultato della richiesta URL è in formato JSON. E' stato quindi necessario salvarlo all'interno di un NSDictionary, tramite un apposito metodo, per poter accedere ai dati.
Il secondo bottone è relativo alla fotocamera. Al click viene aperto l'ImagePicker che, una volta scattata la foto, permette di scegliere se tenerla o scattarne una seconda. In caso si scelga di tenere la foto, l'incarico viene completato e la foto scattata viene salvata nella galleria delle immagini.
Il bottone relativo all'incarico speciale può essere di questi due tipi descritti.

Altra feature permessa da questa classe è il dettaglio dell'incarico al click sulla relativa cella della lista. Il dettaglio lo si può ottenere anche dell'incarico speciale selezionando, questa volta, l'immagine a forma di fulmine che ne caratterizza la presenza.
Il dettaglio consiste in una subview nella quale vi sono titolo, descrizione e punti ottenibili del relativo incarico, nonché un bottone per la chiusura.

CompletatiController

Classe che gestisce una UITableView contenente lo storico degli incarichi ottenuti nei giorni di gioco. Graficamente contiene solo la UITableView e il bottone per tornare alla view principale dell'app.

La lista degli incarichi completati è gestita in modo tale da raggrupparli in base al giorno di ottenimento e ordinare gli stessi gruppi per data, dal più vecchio al più recente. Per questo raggruppamento è stato utilizzato un array di array; l'array principale è servito come raggruppamento di date, e per ciascuna data è stato associato un array contenente tutti gli incarichi di quel giorno.
In particolare ciascuna cella degli incarichi completati comprende una

UILabel nella quale viene specificato il punteggio guadagnato o perso con quel determinato incarico, verde se lo si ha concluso con successo, rosso se lo si ha fallito.
Inoltre, come per la classe sopracitata *InCorsoController*, cliccando sulla cella di un incarico compare una subview contenente titolo, descrizione e punti.

TabBarController

Classe associata al relativo TabBar Controller il cui unico scopo è la personalizzazione grafica della TabBar stessa. Viene cambiato il colore di sfondo, le immagini relative alla selezione o non dei tab e il colore del testo degli stessi.

Database

Probabilmente la classe più importante dell'intera applicazione. Qui dentro sono definiti tutti i metodi che interagiscono con il database SQLite, in modo tale da poterli richiamare da qualsiasi altra classe. Riporto di seguito uno screenshot del file .h contenente la definizione dei metodi e una loro descrizione.

```
//Definisce il percorso del file del database
- (NSString *) filePath;
//Apre il database sfruttando il percorso prima definito
- (sqlite3 *) openDB;
//Al primo avvio della applicazione copia il database nel percorso dei documenti
//di sistema per renderlo modificabile
- (void)copiaDB;

//Restituisce tutti gli incarichi presenti nel database
-(NSMutableArray *)getIncarichiGenerali;
//Restituisce tutti gli incarichi attualmente in corso
-(NSMutableArray*)getIncarichiInCorso;
//Restituisce tutti gli incarichi completati
-(NSMutableArray*)getIncarichiCompletati;
//Restituisce tutti gli incarichi che prevedono l'utilizzo del tocco
-(NSMutableArray*)getIncarichiTocco;
//Restituisce tutte le date nelle quali si ha ricevuto degli incarichi
-(NSMutableArray*)getDateIncarichiCompletati;
//Restituisce tutti i dettagli di un incarico sapendo il suo ID
-(NSMutableDictionary*)findIncaricoById: (id)ID;

//Eliminazione di tutti gli incarichi in corso
-(void)deleteIncarichiInCorso;
//Eliminazione di un incarico dalla tabella di quelli in corso
-(void)deleteIncaricoCompletato: (id)idIncarico;
//Aggiunta di un incarico alla tabella di quelli in corso
-(void)insertIncarichiInCorso: (NSMutableArray*)incarichi :(NSString*)data;
//Aggiunta di tutti gli incarichi falliti nella tabella di quelli completati
-(void)insertIncarichiFalliti: (NSMutableArray*)incarichi;
//Aggiunta di un incarico completato nella omonima tabella
-(void)insertIncaricoCompletato: (NSDictionary*)dictionary :(NSString*)data :(NSString*)special;

//Controllo se vi sono incarichi che necessitano l'utilizzo del GPS
-(BOOL)checkIncarichiGPS;
//Controllo se vi sono incarichi che necessitano l'utilizzo del sensore di prossimità
-(BOOL)checkIncarichiProssimita;
//Controllo se vi sono incarichi che necessitano l'utilizzo dell'accelerometro e il giroscopio
-(BOOL)checkIncarichiMovimento;
//Controllo se vi sono incarichi che necessitano l'utilizzo della fotocamera
-(BOOL)checkIncarichiFoto;
```

3.6 Tabelle del database

Come già spiegato nel paragrafo 3.2 è stato utilizzato un database SQLite utilizzando il plugi-in di Mozilla Firefox SQLite Manager. Vediamo nel dettaglio come sono formate le tabelle utilizzate in questa applicazione.

Incarichi_Generali

Tabella contenente tutti gli incarichi assegnabili all'utente. Non è possibile aggiungerne se non dal lato del programmatore, per cui l'unica maniera per l'utente per ottenerne una maggiore varietà è un aggiornamento del gioco.

Per ogni incarico si ha un ID univoco che lo identifica, il titolo, una descrizione che spiega all'utente cosa deve fare per portare a compimento

l'incarico, i punti ottenibili in caso di successo e detratti in caso di fallimento, una serie di flag per specificare quale sensore è necessario per quel particolare incarico e, infine, nel caso sia necessario il GPS, un campo che specifica la tipologia di zona da ricercare nella richiesta URL.

Column ID	Name	Type	Not Null	Default Value	Primary Key
0	ID	INTEGER	1	null	1
1	Titolo	TEXT	1	null	0
2	Descrizione	TEXT	1	null	0
3	Punti	INTEGER	1	null	0
4	GPS	INTEGER	1	null	0
5	Pros	INTEGER	1	null	0
6	Mov	INTEGER	1	null	0
7	Tocco	INTEGER	1	null	0
8	Foto	INTEGER	0	null	0
9	Zona	TEXT	0	null	0

I campi *GPS*, *Pros*, *Mov*, *Tocco* e *Foto* sono di tipo INTEGER ma sono considerati come BOOL, il loro valore può essere 1 o 0 in base a se sia attivo il relativo sensore per quel dato incarico o meno.

Incarichi_InCorso

Tabella nella quale vengono depositati i cinque incarichi giornalieri. Avendo come puro scopo di essere da appoggio per gli incarichi giornalieri, il suo contenuto molto povero. Comprende unicamente la chiave esterna dalla tabella *Incarichi_Generali* (ID dell'incarico) e la data del giorno in cui è stato ricevuto l'incarico.

Column ID	Name	Type	Not Null	Default Value	Primary Key
0	ID	int	1	null	1
1	Data	text	1	null	0

Incarichi_Completati

Tabella nella quale vengono salvati tutti gli incarichi ricevuti nelle varie giornate di gioco, sia in caso di successo che di fallimento.

Per ogni incarico si ha la chiave esterna dalla tabella *Incarichi_Generali* (ID dell'incarico), la data relativa al giorno in cui si è ricevuto quel dato incarico, un flag per verificare se è stato concluso con successo o fallimento, il punteggio parziale al quale è arrivato l'utente completando l'incarico e un flag per verificare se l'incarico era special o meno.

Column ID	Name	Type	Not Null	Default Value	Primary Key
0	ID	int	1	null	1
1	Data	text	1	null	2
2	Concluso	INTEGER	0	null	0
3	Punteggio	INTEGER	0	null	0
4	Special	INTEGER	0	null	0

Come nella tabella degli incarichi generali i flag “Concluso” e “Special” sono INTEGER ma sono utilizzati come BOOL.

Parlando in generale di tutte e tre le tabelle, è stato scelto di utilizzare i campi relativi alla data sotto forma di testo con una particolare formattazione.
In questa maniera, nel codice, per trasformare la data da formato stringa a formato NSDate è stato necessario fare un parsing definendo la sintassi nella quale era scritta la data. Tutto questo è stato possibile grazie al tipo di dato NSDateFormatter.

```
NSDateFormatter* dateFormatter = [[NSDateFormatter alloc] init];
[dateFormatter setDateFormat:@"dd/MM/yyyy HH:mm"];
NSDate *data;
data = [dateFormatter dateFromString:[[incarichi objectAtIndex:0] objectForKey:@"Data"]];
```

In questo caso la formattazione della data è stata di tipo giorno/mese/anno ora/minuti.

3.7 Problematiche implementative e risoluzione

Il più grande problema di implementazione riscontrato riguarda l'accesso ai metodi della classe *Database*. Durante alcune fasi di debug ci sono stati

degli errori nel passaggio degli incarichi da “in corso” a “completato”. Effettuando alcune prove si è dedotto che il problema era di concorrenza in quanto, in alcuni cicli, si accedeva ai metodi di scrittura e lettura da database tramite la stessa istanza della classe, provocando collisioni di dati.

Il problema è stato risolto istanziando un nuovo oggetto *Database* ogni qualvolta era necessario accedervi, annullando la collisione.

Altri problemi sono stati causati dal controllo per l'assegnazione di nuovi incarichi dopo il mezzogiorno del giorno successivo. Per evitare che venissero assegnati due volte nello stesso giorno i cinque incarichi, si è fatto ricorso ad un metodo di confronto di giorni. Nel momento in cui fossero passate le 12 e il giorno fosse stato diverso dall'ultimo nel quale si erano ricevuti incarichi allora ne sarebbero stati assegnati ulteriori cinque. Operando in questa maniera, però, nel momento in cui l'utente avesse ottenuto, ad esempio, gli incarichi alle 11, una volta passate le 12 non glieli avrebbe riassegnati in quanto il giorno sarebbe stato lo stesso.
Per ovviare a questa problematica si è deciso di salvare la data di ottenimento degli incarichi decrementandola di un giorno nel caso si fossero ottenuti prima delle 12.

Capitolo 4

CONCLUSIONI E SVILUPPI FUTURI

In questa tesi si è cercato di sviluppare un'applicazione per dispositivi mobili che consenta l'interazione con una fantomatica morosa esigente con lo scopo di assolvere agli incarichi assegnati da essa per migliorarne il rapporto e poterla sposare raggiunto il punteggio massimo.

Inizialmente è stata effettuata una ricerca negli store digitali per verificare come fossero applicazioni dello stesso genere in modo tale da proporre un prodotto che prenda spunto da essi ma con uno stile di gioco completamente diverso. E' stata poi fatta una valutazione su quali strumenti utilizzare per la realizzazione dell'applicazione considerando le conoscenze e la difficoltà di ciascuno.
L'aspetto nel quale si è creduto di più è l'auto-conclusione del gioco una volta raggiunto il punteggio massimo.
Stiamo vivendo in un periodo storico nel quale i giochi, in particolare le applicazioni, sono continuamente aggiornati per dare continua longevità all'utilizzo che ne fa un utente medio. Se, dal punto di visto della passione verso un particolare titolo, può essere giusto aggiungere contenuti per “convincere” i giocatori a continuare ad utilizzare una determinata applicazione, dall'altro si è perso nel tempo il gusto dell'averla conclusa.
Per questo motivo è stato scelto di dare una conclusione al gioco, evitando di lasciare l'applicazione nel dimenticatoio per la troppa continuità, che spesso porta alla monotonia.

SVILUPPI FUTURI

Gli sviluppi futuri per questa applicazione possono essere tantissimi. Il fatto di aver proposto una versione molto basilare e gestionale permette di stravolgerla completamente e renderla molto più elastica.

In primis **Kibi** stessa. Per la realizzazione di questa tesi è stato scelto di disegnare la protagonista in una determinata maniera, scegliendo i colori per fare di lei una specie di stereotipo. Uno sviluppo futuro potrebbe essere di dare la possibilità all'utente, al primo avvio, di scegliere tra varie tipologie di Kibi tra cui scegliere con capelli, occhi e viso diversi.

In parallelo all'estetica della protagonista è possibile aggiungere svariate nuove **animazioni** per rendere maggiormente partecipe e viva l'interazione con il giocatore.

Per quanto riguarda gli **incarichi**, oltre ad aggiungerne molti di più e con caratteristiche diverse, è possibile particolareggiarli. Ad esempio, nel caso in cui Kibi assegni l'incarico di ballare, sarebbe possibile specificare una determinata sequenza di movimenti da dover fare con lo smartphone per portare a compimento l'incarico.

Altro aspetto importante, sempre riguardante gli incarichi, è sicuramente il **check-in** tramite GPS per gli incarichi che lo richiedono. In questa versione base il check-in viene effettuato manualmente utilizzando il relativo bottone. Una miglioria sarebbe potrebbe essere controllare in background la propria località per avvisare l'utente del completamento dell'incarico qualora si sia giunti in prossimità della zona richiesta.

Ultima componente potrebbe essere la realizzazione di maggiori **notifiche** mandate all'utente per ricordargli che deve ancora completare un determinato numero di incarichi.

BIBLIOGRAFIA

- iPhone, Wikipedia, *http://it.wikipedia.org/wiki/IPhone*

- App Store, Wikipedia, *http://it.wikipedia.org/wiki/App_Store*

- Xcode, Wikipedia, *http://it.wikipedia.org/wiki/Xcode*

- Xcode, Apple Developer, *https://developer.apple.com/xcode/*

- SQLite, Wikipedia, *http://it.wikipedia.org/wiki/SQLite*

- SQLite, sito ufficiale, *http://www.sqlite.org/*

- Adobe Photoshop, Wikipedia, *http://it.wikipedia.org/wiki/Adobe_Photoshop*

- Core Motion Framework Reference, iOS Developer Library, *https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CoreMotion_Reference/_index.html*

- Core Location Framework Reference, iOS Developer Library, *https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CoreLocation_Framework/_index.html*

- Facebook SDK, Facebook Developers, *https://developers.facebook.com/docs/ios/*

- Twitter SDK, Twitter Developers, *https://dev.twitter.com/*